普通高等教育“十一五”国家级规划教材

“信息化与信息社会”系列丛书之
高等学校信息安全专业系列教材

信息系统安全概论

石文昌　梁朝晖　编著
沈昌祥　审

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书以独特的方式讲授以主机为中心的系统安全的基本思想、技术和方法。本书的宗旨是帮助读者认识每个人手中、家里、工作单位中、甚至庞大的数据处理中心深处的计算机主机系统的安全问题及其解决途径。

本书由三部分组成，第一部分是基础篇，包含第 1～3 章，内容包括信息系统安全绪论、计算机系统基础和可信计算平台基础等；第二部分是核心篇，包含第 4～8 章，内容包括操作系统的基础安全性、操作系统的增强安全性、数据库系统的基础安全性、数据库系统的增强安全性和系统完整性保护等；第三部分是拓展篇，包含第 9～10 章，内容包括基于主机的入侵检测和计算机病毒原理及其防治等。

本书的特色是透过信息网络空间的宏观安全体系结构去看待主机系统的安全问题，通过主机系统与信息安全知识体系的融合去认识主机系统的安全问题，采取核心硬件、系统软件和应用软件相结合的综合手段去分析主机系统的安全问题，运用安全性与可信性有机统一的整体措施去解决主机系统的安全问题，同时，本书注意体现网络环境对系统安全的影响及系统安全对整体安全的支撑。

本书可作为高等学校计算机、信息安全、电子与通信及相关专业的本科生和研究生的教材或参考书，也可供从事相关专业教学、科研和工程技术的人员参考。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

信息系统安全概论/石文昌，梁朝晖编著. —北京：电子工业出版社，2009.3
（信息化与信息社会系列丛书）
普通高等教育“十一五”国家级规划教材．高等学校信息安全专业系列教材
ISBN 978-7-121-08222-1

Ⅰ.信… Ⅱ.①石… ②梁… Ⅲ. 信息系统—安全技术—高等学校—教材 Ⅳ.TP309

中国版本图书馆 CIP 数据核字（2009）第 013611 号

策划编辑：刘宪兰
责任编辑：张 京
印 刷：
装 订：
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1092 1/16 印张：30.25 字数：644 千字
印 次：2009 年 3 月第 1 次印刷
印 数：4 000 册 定价：46.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。
服务热线：（010）88258888。

作者简介



石文昌

教授，博士生导师。北京大学计算机科学与技术理学学士，中国科学院计算机软件工学硕士和计算机软件与理论工学博士。中国科学院院长奖获得者，红旗安全操作系统总设计师。现任中国人民大学教授，兼中国科学院研究生院教授，中国人民大学系统与信息安全研究实验室负责人，中国计算机学会高级会员，中国计算机学会信息保密、系统软件、开放系统等专委会委员，IEEE 会员。曾任中国科学院软件所研究员。长期从事系统软件与信息安全的研究、开发与人才培养工作，1997 年晋升为教授，承担了包括国家 863 高技术研究发展计划项目和国家自然科学基金项目在内的一系列重要科研项目。1988 年开始进行操作系统的研究，1991 年成功完成 UNIX 操作系统的移植，为以 UNIX 为基础的自主操作系统的开发做出了重要贡献，成果获国家科技攻关重大成果奖。2001 年主持研制成功通过国家标准 GB 17859 第三级认定的红旗安全操作系统，在产业应用中产生了广泛影响。2004 年主持完成了国家 863 计划信息安全技术主题重点课题，解决了国家标准 GB 17859 第四级安全操作系统和安全数据库管理系统研制中的一系列关键技术问题，为高安全、高可信系统软件的研制打下了良好的基础。在理论创新方面，证明了经典 BLP 安全模型的动态适应性能力，证明了多级安全策略模型的历史敏感性特征，纠正了国际同行在多级安全性策略特性研究方面的错误结论；另外，基于主观逻辑理论，提出了信息安全国际 CC 标准的定量评价模型，该工作在推动 CC 标准的应用及信任体系量化模型的研究方面具有重要的基础性意义。



梁朝晖

博士，现任中国人民大学信息学院讲师。1989 年获华中科技大学工学学士学位，1992 年获华中科技大学工学硕士学位，2005 年获北京邮电大学博士学位。研究方向：信息安全与宽带无线网络。作为第一参加人或第二参加人参加国家 863 项目 2 项，国家自然基金项目 3 项，北京自然科学基金项目 1 项。近期在国内外发表论文 12 篇，其中已有 5 篇均为第一作者的论文被 EI 检索。以第一申请人申请专利 1 项。

总　　序

信息化是世界经济和社会发展的必然趋势。近年来，在党中央、国务院的高度重视和正确领导下，我国信息化建设取得了积极进展，信息技术对提升工业技术水平、创新产业形态、推动经济社会发展发挥了重要作用。信息技术已成为经济增长的“倍增器”、发展方式的“转换器”、产业升级的“助推器”。

作为国家信息化领导小组的决策咨询机构，国家信息化专家咨询委员会一直在按照党中央、国务院领导同志的要求就信息化前瞻性、全局性和战略性的问题进行调查研究，提出政策建议和咨询意见。在做这些工作的过程中，我们愈发认识到，信息技术和信息化所具有的知识密集的特点，决定了人力资本将成为国家在信息时代的核心竞争力，大量培养符合中国信息化发展需要的人才已成为国家信息化发展的一个紧迫需求，成为我国应对当前严峻经济形势，推动经济发展方式转变，提高在信息时代参与国际竞争比较优势的关键。2006 年 5 月，我国《2006—2010 年国家信息化发展战略》公布，提出“提高国民信息技术应用能力，造就信息化人才队伍”是国家信息化推进的重点任务之一，并要求构建以学校教育为基础的信息化人才培养体系。

为了促进上述目标的实现，国家信息化专家咨询委员会一直致力于通过讲座、论坛、出版物等各种方式推动信息化知识的宣传、教育和培训工作。2007 年，国家信息化专家咨询委员会联合教育部、原国务院信息化工作办公室成立了“信息化与信息社会”系列丛书编委会，共同推动“信息化与信息社会”系列丛书的组织编写工作。编写该系列丛书的目的，是力图结合我国信息化发展的实际和需求，针对国家信息化人才教育和培养工作，有效梳理信息化的基本概念和知识体系，通过高校教师、信息化专家、学者与政府官员之间的相互交流和借鉴，充实我国信息化实践中的成功案例，进一步完善我国信息化教学的框架体系，提高我国信息化图书的理论

和实践水平。毫无疑问，从国家信息化长远发展的角度来看，这是一项带有全局性、前瞻性和基础性的工作，是贯彻落实国家信息化发展战略的一个重要举措，对于推动国家的信息化人才教育和培养工作，加强我国信息化人才队伍的建设具有重要意义。

考虑当前国家信息化人才培养的需求、各个专业和不同教育层次（博士生、研究生、本科生）的需要，以及教材开发的难度和编写进度时间等问题，“信息化与信息社会”系列丛书编委会采取了集中全国优秀学者和教师、分期分批出版高质量的信息化教育丛书的方式，根据当前高校专业课程设置情况，先开发“信息管理与信息系统”、“电子商务”、“信息安全”三个本科专业高等学校系列教材，随后再根据我国信息化和高等学校相关专业发展的情况陆续开发其他专业和类别的图书。

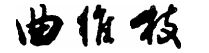
对于新编的三套系列教材（以下简称系列教材），我们寄予了很大希望，也提出了基本要求，包括信息化的基本概念一定要准确、清晰，既要符合中国国情，又要与国际接轨；教材内容既要符合本科生课程设置的要求，又要紧跟技术发展的前沿，及时地把新技术、新趋势、新成果反映在教材中；教材还必须体现理论与实践的结合，要注意选取具有中国特色的成功案例和信息技术产品的应用实例，突出案例教学，力求生动活泼，达到帮助学生学以致用的目的，等等。

为力争出版一批精品教材，“信息化与信息社会”系列丛书编委会采用了多种手段和措施保证系列教材的质量。首先，在确定每本教材的第一作者的过程中引入了竞争机制，通过广泛征集、自我推荐和网上公示等形式，吸收优秀教师、企业人才和知名专家参与写作；其次，将国家信息化专家咨询委员会有关专家纳入到各个专业编委会中，通过召开研讨会和广泛征求意见等多种方式，吸纳国家信息化一线专家、工作者的意见和建议；再次，要求各专业编委会对教材大纲、内容等进行严格的审核，并对每一本教材配有一至两位审稿专家。

如今，我们很高兴地看到，在教育部和原国务院信息化工作办公室的支持下，通过许多高校教师、专家学者及电子工业出版社的辛勤努力和付出，“信息化与信息社会”系列丛书中的三套系列教材即将陆续和读者见面。

我们衷心期望，系列教材的出版和使用能对我国信息化相应专业领域的教育发展和教学水平的提高有所裨益，对推动我国信息化的人才培养有所贡献。同时，我们也借系列教材开始陆续出版的机会，向所有为系列教材的组织、构思、写作、审核、编辑、出版等做出贡献的专家学者、老师和工作人员表达我们最真诚的谢意！

应该看到，组织高校教师、专家学者、政府官员以及出版部门共同合作，编写尚处于发展动态之中的新兴学科的高等学校教材，还是一个初步的尝试。其中，固然有许多的经验可以总结，也难免会出现这样那样的缺点和问题。我们衷心地希望使用系列教材的教师和学生能够不吝赐教，帮助我们不断地提高系列教材的质量。



2008 年 12 月 15 日

序　　言

人类走过了农业社会、工业社会，如今正处于信息社会的伟大时代，“信息社会”这个词语无疑已经家喻户晓，信息化的大潮正席卷着世界的每一个角落。地球两端，万里之隔，人们能通过互联网与亲朋畅快交流，音容笑貌犹如就在眼前，真正是天涯变咫尺；分支机构遍布全球的庞大企业运转有条不紊，各机构协作顺畅，其功能强大的信息系统功勋卓著；分析复杂神秘的生物基因，预测瞬息万变的天气趋势，有了容量惊人的数据库系统和“聪明绝顶”的高性能计算系统，科学家们如虎添翼。总之，人类处处受益于信息化成果并正在信息化这条大道上加速前进，决不会放慢脚步。

然而，阳光之下总会有阴影，人类越依赖于信息系统，信息安全问题就越发凸显。关于信息安全的形形色色的新闻日益频繁地见诸于媒体：某银行数据库数据被窃取导致客户信息泄露，使客户惶惶不安，银行面临信任危机；某计算机病毒大肆泛滥，无数用户系统瘫痪，让相关企业损失惨重；某国军方网络被黑客侵入，军事机密竟被人如探囊取物般轻易窃取……这样的事件一再提示我们，信息安全问题是社会信息化发展进程中无法回避的客观产物，只有主动积极地面对和解决这一问题才能保障信息化的顺利推进，确保经济、社会的稳定乃至国家的安全。

目前，世界各国政府在信息安全领域的重视程度正在不断加大，并纷纷推出了本国的相关标准、规范或法律，大力扶持高校和其他科研机构对信息安全问题的研究，同时采取各种措施促进信息安全领域的人才培养以满足本国信息化建设的需要，为本国的信息产业发展提供中坚力量。特别是一些信息化进程起步较早，水平较高的发达国家，其信息安全领域的研究水平和产业化程度已相当令人瞩目。

我国正处于信息化建设的关键阶段，2006 年发布的《2006—2010 年国家信息化发展战略》更是从战略的高度指出了推进信息化对我国经济建设和国家发展的重要作用，规划出了新时期我国信息化发展的宏伟蓝图。由此可见，我国的信息化建设和信息产业正面

临前所未有的机遇和挑战。

正是在这样的时代背景下，信息安全问题越来越引起全社会上下的广泛关注。信息安全领域必须不断提高研究水平以满足经济建设和国家安全的需要，为我国信息化建设的大踏步前进保驾护航，为创建和谐社会，实现可持续发展贡献力量。因此，大量高素质的信息安全人才成为了最急需、最宝贵的资源。

康有为曾经说过：“欲任天下之事，开中国之新世界，莫亟于教育”。我们的国家要想不断发展科技，增强国力，开创出我们自己富强文明的“新世界”，必须加大力度进行信息化建设。而要使我国的信息化水平走在世界前列，全面提高信息安全领域教育水平，特别是促进高等学校信息安全专业对相关人才的培养和教育，就成为了成败的关键。高等学校信息安全系列教材的编撰就是希望能够为我国的信息安全领域专业人才的培养、为我国信息化水平的腾飞助一臂之力。

信息安全专业教育有其自身的特点，要求学习该专业的学生能够将系统知识与专业知识有机结合，在注重提升理论高度的同时还要能够把理论知识与工程实践紧密联系起来。本系列教材针对高等学校信息安全专业教育的这些特点，同时根据其知识体系、教育层次和课程设置，规划了教材的内容，增加了实际案例，力争做到既紧跟前沿技术的发展，又不失扎实的基本理论和生动活泼的形式，使学生能够学以致用。本系列教材从不同角度论述和总结了信息安全领域的科学问题，有着较强的适用性，既可作为高等学校信息安全专业和相关专业本科生的教材，也可以作为非信息安全专业的公共教科书，同时还可以作为从事信息安全工作的科研技术人员和管理人员的培训教材或参考书，使其了解信息安全相关关键技术和发展态势。

信息安全科学在不断发展，我们也将会努力使本系列教材适应和紧跟这种发展的节奏，使我们培养的信息安全人才能够与时俱进，用自己的所学共筑我国信息安全的万里长城。

限于作者的水平，本系列教材难免存在不足之处，敬请读者批评指正。

高等学校信息安全专业系列教材编委会

2008 年 10 月

前　言

正如人是现实社会生活中的行为单元一样，主机是信息网络空间中的工作单元。为了维护现实社会生活的安定与和谐，公安机关采取了对犯罪分子进行严厉打击的措施，为了确保信息网络空间的安全与可信，有必要对主机系统的安全问题准确地把握。

信息安全事关国家安全，因为信息化已经渗透到人类社会的各个层面。主机系统的安全在信息安全中举足轻重，因为所有的软件最终都必须落实到具体的主机系统上运行。解剖信息网络空间中的安全问题，就能清楚地看到主机系统安全是其中不可或缺的成分，而解剖主机系统的安全问题，就会发现它自身的内容也是丰富多彩的。

本书重点讲授以主机为中心的系统安全的基本思想、技术和方法。值得一提的是：这里所说的主机并非等同于 20 世纪 60～70 年代盛行的大型主机，而是指包含 21 世纪的终端、个人计算机、工作站和服务器等的各种计算机设备。正如人不是孤立的人而是社会中的人一样，这里所谈论的主机也绝不是孤立的主机，而是处在开放的网络互联环境中的主机。

本书的内容设计宗旨是帮助读者认识每个人手中、家里、工作单位中、甚至庞大的数据处理中心深处的计算机主机系统的安全问题及其解决途径。信息安全是一个复杂的系统工程，主机系统安全是其中的一个重要环节，而且，这个环节与用户息息相关。学习主机系统安全对营造用户身边计算机的安全环境大有裨益。

在信息网络空间的信息安全体系这个层面上，主机系统安全是其中的一个关键点。本书系统地讲授这个点的关键思想，以期帮助读者把握这个点的基本内在机理，并了解这个点在整个面上的地位和作用。本书的编写理念是以点为目标，注意点面结合。学习主机系统安全对于建设整个信息网络空间的安全氛围也一定很有帮助。

本书的特色是透过信息网络空间的宏观安全体系结构去看待主机系统的安全问题；通过主机系统与信息安全知识体系的融合去认识主机系统的安全问题；采取核心硬件、系统软件和应用软件相结

合的综合手段去分析主机系统的安全问题；运用安全性与可信性有机统一的整体措施去解决主机系统的安全问题。同时，本书注意体现网络环境对系统安全的影响及系统安全对整体安全的支撑。

本书是在总结编者多年从事系统软件与信息安全的科研实践和人才培养工作的基础上编写而成的。在编写过程中，考虑到了人才培养和人才需求方面的一些实际情况。

鉴此，在内容的选择和编排上，力求知识的系统性，但不求内容的全面性，尤其注重选材的典型性及其应用价值。例如，在安全模型方面，本书并没有列出太多的安全模型，只是有选择地扼要地介绍了其中有限的几个，并且，有意识地在介绍系统实现时尽量体现所介绍的安全模型的应用方法，希望以此来帮助读者更好地理解相应的安全模型，并达到举一反三的效果。

本书的内容可以分为三个部分。第一部分为基础篇，介绍系统安全的必备基础知识，由第 1～3 章构成；第二部分为核心篇，介绍系统安全的核心内容，由第 4～8 章构成；第三部分为拓展篇，介绍向应用推进的系统安全内容，由第 9～10 章构成。

信息安全基本认识是系统安全课程的开端，是学习第 1 章要达到的主要目的。计算机系统基础是建立系统安全思想的根基，第 2 章通过回顾计算机硬件、操作系统和数据库系统等方面的基本内容来巩固。第 3 章介绍的可信计算平台通过安全芯片提供基本的安全功能，可以作为系统安全的硬件基础。

操作系统安全性和数据库系统安全性是系统安全的核心内容，值得分别从基础安全性和增强安全性两个层面去把握，第 4～7 章的篇幅专门为此目的进行设置。第 8 章介绍的系统完整性保护是系统安全核心内容的另一个重要方面，它不但可以体现在操作系统安全性和数据库系统安全性中，还可以体现在硬件的安全支持和应用系统的安全需求中。

操作系统和数据库系统在系统安全中处于核心地位，其核心意义主要体现在它们对于确保应用系统的安全性上具有不可或缺的重要作用。应用系统的安全性是用户希望实现的根本目标。基于主机的入侵检测和计算机病毒原理及其防治是系统安全由核心层向应用层延伸的重要内容，分别在第 9 章和第 10 章介绍。

如前所述，本书不求内容的全面性，但求知识的系统性。我们着力给读者讲授计算机主机系统安全的统一知识体系，引导读者领略系统安全知识框架的整体概貌，掌握系统安全的基础知识和关键

技术，为读者学习信息安全知识、掌握信息安全技术、解决信息安全问题及进一步从网络安全等其他方面充实信息安全学识打下坚实的基础。

本书的编著工作得到了国家 863 高技术研究发展计划课题（2007AA01Z414）和国家自然科学基金项目（60373054，60703102，60703103）的资助，在此，我们向国家的相关机构表示衷心的感谢。

在本书的编著过程中，我们参考了大量的技术文献、著作和教材，这些文献、著作和教材的作者的智慧结晶和出版机构的贡献使我们受益匪浅，为本书的编写奠定了宝贵的基础，在此，我们向相关作者和出版机构致以崇高的敬意和诚挚的谢意。

由于我们的学识和水平有限，书中难免有错误和不妥之处，敬请读者批评指正。关于本书的任何问题，都欢迎通过下面的电子邮件与我们联系。同时，希望本书能为信息安全教育做出新的贡献。

电子邮件：syssecbook(at)gmail.com

石文昌

2008 年 11 月

中国人民大学

目　　录

第一部分　基　础　篇

第 1 章　信息系统安全绪论 ····· 3

1.1　安全攻击实景呈现 ····· 4

1.1.1　诱惑及初探 ····· 4

1.1.2　确定合适的突破口 ····· 5

1.1.3　设法扩大战果 ····· 7

1.1.4　全面出击 ····· 7

1.1.5　尾声 ····· 8

1.2　安全攻击环节概览 ····· 9

1.2.1　侦察 ····· 9

1.2.2　扫描 ····· 10

1.2.3　获取访问 ····· 11

1.2.4　维持访问 ····· 12

1.2.5　掩盖踪迹 ····· 13

1.3　信息安全典型事件 ····· 13

1.4　信息安全经典要素 ····· 15

1.4.1　机密性 ····· 15

1.4.2　完整性 ····· 16

1.4.3　可用性 ····· 17

1.5　信息系统安全策略 ····· 18

1.5.1　信息安全威胁 ····· 18

1.5.2　策略与机制 ····· 20

1.5.3　安全的目的 ····· 21

1.5.4　安全策略的意义 ····· 22

1.5.5　安全策略的类型 ····· 25

1.6　信息系统访问控制 ····· 27

1.6.1　访问控制矩阵 ····· 27

1.6.2　访问控制的类型 ····· 29

1.6.3　贝尔-拉普杜拉访问控制模型 ····· 30

1.6.4　系统保护状态 ····· 32

1.6.5　访问控制结构与设计原则 ····· 33

1.7　系统安全知识定位 ····· 35

1.7.1 系统安全的宏观定位 …… 35
1.7.2 系统安全的知识点定位 …… 36
1.8 本章小结 …… 37
习题 1 …… 38
第 2 章 计算机系统基础 …… 41
2.1 程序员眼中的计算机系统 …… 42
2.1.1 计算机系统的硬件组成 …… 43
2.1.2 执行 hello 程序 …… 45
2.1.3 高速缓存 …… 46
2.1.4 层次结构的存储设备 …… 47
2.1.5 操作系统管理硬件 …… 48
2.1.6 通过网络与其他系统通信 …… 51
2.2 计算机组成基础 …… 52
2.2.1 中央处理单元 …… 52
2.2.2 主存储器 …… 53
2.2.3 输入/输出子系统 …… 56
2.2.4 子系统的内部连接 …… 57
2.3 操作系统基础 …… 59
2.3.1 操作系统的发展及其意义 …… 60
2.3.2 操作系统的演化 …… 62
2.3.3 操作系统的构成 …… 63
2.3.4 现代操作系统的特征 …… 68
2.3.5 Linux 和 Windows 操作系统结构 …… 70
2.4 数据库系统基础 …… 74
2.4.1 数据模型 …… 74
2.4.2 概念模型 …… 76
2.4.3 关系模型 …… 78
2.4.4 数据库系统结构 …… 82
2.4.5 数据库系统的组成 …… 85
2.5 本章小结 …… 88
习题 2 …… 89
第 3 章 可信计算平台基础 …… 91
3.1 可信计算发展概貌 …… 92
3.1.1 TCG 可信计算的典型前期基础 …… 92
3.1.2 TCG 可信计算的发展思路 …… 93
3.1.3 响应 TCG 规范的热点研究 …… 94
3.2 可信计算平台的基本特性 …… 95
3.2.1 保护能力 …… 95

3.2.2 对外证明 …… 96
3.2.3 完整性度量、存储和报告 …… 96
3.3 可信计算平台的基本体系 …… 97
3.3.1 平台的可信构件块 …… 97
3.3.2 信任边界 …… 97
3.3.3 信任传递 …… 98
3.3.4 完整性度量 …… 98
3.3.5 完整性报告 …… 99
3.3.6 以 TPM 为通信端点 …… 104
3.3.7 存储保护 …… 106
3.4 可信平台模块 …… 109
3.4.1 TPM 的组件 …… 109
3.4.2 通信接口 …… 110
3.4.3 具有篡改保护能力的装配 …… 110
3.5 可信计算平台的隐私问题 …… 110
3.6 可信计算平台的运行模型 …… 111
3.6.1 TPM 的工作状态 …… 111
3.6.2 平台的工作方法 …… 114
3.6.3 平台的软件接口 …… 115
3.6.4 TPM 命令的授权验证 …… 120
3.7 可信计算平台的编程接口 …… 124
3.7.1 编程相关的 TCG 命名习惯 …… 125
3.7.2 程序员视角的 TPM 结构 …… 125
3.7.3 TPM 的启动与清零 …… 127
3.7.4 在程序中使用 TPM 命令 …… 127
3.7.5 TPM 命令的基本用途 …… 131
3.8 本章小结 …… 133
习题 3 …… 134

第二部分 核 心 篇

第 4 章 操作系统的基础安全性 …… 139
4.1 操作系统安全概貌 …… 140
4.1.1 操作系统安全简史 …… 140
4.1.2 操作系统安全的主要内容 …… 142
4.1.3 必不可少的操作系统安全性 …… 142
4.2 身份标识与认证的基本方法 …… 147
4.2.1 身份标识的基本方法 …… 147
4.2.2 身份认证的基本方法 …… 149

4.2.3 口令信息的管理方法 …… 151
4.3 面向网络的身份认证 …… 157
4.3.1 认证信息的网络化管理 …… 157
4.3.2 认证信息的加密传输 …… 159
4.3.3 面向服务的再度认证 …… 161
4.4 基于 PAM 的统一认证框架 …… 163
4.5 基于权限位的访问控制 …… 165
4.5.1 访问权限的定义与表示 …… 165
4.5.2 用户的划分与访问控制 …… 167
4.5.3 访问控制算法 …… 168
4.6 进程的有效身份与权限 …… 170
4.6.1 进程与文件和用户的关系 …… 170
4.6.2 进程的用户属性 …… 171
4.6.3 进程有效用户属性的确定 …… 172
4.7 基于 ACL 的访问控制 …… 175
4.7.1 ACL 的表示方法 …… 175
4.7.2 基于 ACL 的访问判定 …… 177
4.8 特权分割与访问控制 …… 179
4.8.1 特权的意义与问题 …… 179
4.8.2 特权的定义 …… 179
4.8.3 基于特权的访问控制 …… 181
4.9 加密文件系统 …… 182
4.9.1 加密文件系统的应用方法 …… 182
4.9.2 加密文件系统的基本原理 …… 186
4.9.3 加密算法的加密密钥 …… 187
4.10 系统行为审计 …… 189
4.10.1 审计机制的结构 …… 190
4.10.2 审计指令的配置 …… 191
4.10.3 审计信息的分析 …… 193
4.11 本章小结 …… 194
习题 4 …… 196
第 5 章 操作系统的增强安全性 …… 199
5.1 TE 模型与 DTE 模型 …… 200
5.1.1 TE 模型的基本思想 …… 200
5.1.2 DTE 模型的基本思想 …… 203
5.2 SELinux 实现的 TE 模型 …… 206
5.2.1 SETE 模型与 DTE 模型的区别 …… 206
5.2.2 SETE 模型的访问控制方法 …… 207

5.2.3 授权进程切换工作域 …… 208
5.2.4 进程工作域的自动切换 …… 212
5.3 访问判定与切换判定 …… 213
5.3.1 SELinux 的访问判定 …… 213
5.3.2 SELinux 的切换判定 …… 215
5.3.3 客体类型标签的存储 …… 218
5.4 SELinux 的系统结构设计 …… 219
5.4.1 Linux 安全模块框架 …… 219
5.4.2 SELinux 内核体系结构 …… 220
5.4.3 用户空间的客体管理器 …… 223
5.5 SELinux 的策略语言 …… 225
5.5.1 SEPL 策略源文件及其编译 …… 226
5.5.2 安全策略的构造与装载 …… 227
5.5.3 策略源模块样例 …… 229
5.6 本章小结 …… 231
习题 5 …… 233
第 6 章 数据库系统的基础安全性 …… 235
6.1 数据库系统安全概貌 …… 236
6.1.1 数据库安全的经典要素观 …… 236
6.1.2 数据库安全的典型研究课题 …… 238
6.2 关系数据库自主访问授权 …… 241
6.2.1 授权的发放与回收 …… 241
6.2.2 否定式授权 …… 244
6.2.3 可选的授权回收方式 …… 248
6.2.4 授权的时效性 …… 251
6.2.5 系统级的访问授权 …… 254
6.3 基于视图的访问控制 …… 256
6.3.1 基于内容的访问控制需求 …… 256
6.3.2 基于视图的读访问控制 …… 256
6.3.3 基于视图的写访问控制 …… 258
6.3.4 视图机制的作用和不足 …… 260
6.4 基于角色的访问控制 …… 261
6.4.1 RBAC 的基本思想 …… 261
6.4.2 RDBMS 中的 RBAC …… 263
6.4.3 角色授权与非递归式授权回收 …… 265
6.5 数据库数据的推理控制 …… 266
6.5.1 数据库数据的推理方法 …… 267
6.5.2 数据库数据的推理控制 …… 272

6.6 本章小结 275
习题 6 277
第 7 章 数据库系统的增强安全性 279
7.1 虚拟专用数据库机制 280
7.1.1 初识 VPD 机制 280
7.1.2 VPD 机制的工作原理 283
7.1.3 基于访问类型的控制的实施 285
7.1.4 VPD 安全防线 290
7.1.5 免受 VPD 机制控制的措施 291
7.1.6 面向敏感字段的 VPD 功能 292
7.2 基于标签的安全机制 295
7.2.1 标签的基本构成 295
7.2.2 基于标签的数据库访问控制 297
7.2.3 基于标签的安全机制的实现 298
7.2.4 OLS 安全策略的创建 299
7.2.5 标签的等级元素的创建与应用 301
7.2.6 标签的类别元素的创建与应用 307
7.2.7 标签的组别元素的创建与应用 313
7.2.8 借助会话标签给记录标签赋值 318
7.2.9 再谈基于标签的授权 319
7.3 本章小结 325
习题 7 327
第 8 章 系统完整性保护 329
8.1 完整性的经典模型 330
8.1.1 毕巴模型 330
8.1.2 克拉克-威尔逊模型 334
8.1.3 莫科尔树模型 338
8.2 基于系统安全引导的完整性 341
8.2.1 系统引导的一般过程 341
8.2.2 系统的可信引导过程 343
8.2.3 组件完整性验证技术 344
8.2.4 AEGIS 模型的安全引导过程 345
8.3 基于安全协处理器的完整性 347
8.3.1 IBM 4758 安全协处理器的硬件组织结构 348
8.3.2 IBM 4758 安全协处理器的软件层次结构 350
8.3.3 可信代码的安装与更新 351
8.3.4 系统的可信引导 354
8.3.5 系统完整性保障措施 355

8.4 基于安全中央处理器的完整性 …… 355
8.4.1 进程完整性验证框架 …… 355
8.4.2 完整性验证单元 …… 358
8.4.3 硬件支持的完整性验证 …… 360
8.5 内核主导的完整性度量 …… 361
8.5.1 完整性度量对象的构成 …… 362
8.5.2 完整性度量的基本机制 …… 364
8.5.3 完整性度量的实现方法 …… 366
8.6 文件系统的完整性检查 …… 369
8.6.1 Tripwire 的原理与组成 …… 369
8.6.2 Tripwire 的工作模式 …… 372
8.6.3 完整性检查策略的定义 …… 372
8.6.4 Tripwire 的基本用法 …… 376
8.7 本章小结 …… 377
习题 8 …… 379

第三部分 拓 展 篇

第 9 章 基于主机的入侵检测 …… 383
9.1 入侵检测概述 …… 384
9.1.1 入侵检测的概念 …… 384
9.1.2 入侵检测系统模型 …… 385
9.1.3 入侵检测系统的作用 …… 386
9.1.4 入侵检测系统的分类 …… 387
9.2 基于主机入侵检测技术的发展历程及发展趋势 …… 390
9.2.1 发展历程 …… 390
9.2.2 发展趋势 …… 391
9.3 基于主机入侵检测系统的信息获取 …… 392
9.3.1 操作系统的审计记录 …… 392
9.3.2 系统日志 …… 396
9.3.3 应用程序的日志文件 …… 397
9.4 基于主机的滥用入侵检测方法 …… 398
9.4.1 操作系统层的滥用入侵检测方法 …… 399
9.4.2 应用程序层的滥用入侵检测方法 …… 401
9.5 基于主机的异常入侵检测方法 …… 402
9.5.1 操作系统层的异常入侵检测方法 …… 402
9.5.2 应用程序层的异常入侵检测方法 …… 404
9.6 文件完整性检查 …… 405
9.7 本章小结 …… 406

习题 9 …… 406
第 10 章　计算机病毒原理及其防治 …… 409
10.1　计算机病毒的概念 …… 410
10.2　计算机病毒的发展历程及发展趋势 …… 411
10.2.1　计算机病毒的发展历程 …… 411
10.2.2　计算机病毒的发展阶段 …… 415
10.2.3　计算机病毒的发展趋势 …… 416
10.3　计算机病毒的表现形式 …… 417
10.4　计算机病毒的特征 …… 420
10.4.1　计算机病毒的一般特征 …… 420
10.4.2　计算机病毒的新特征 …… 424
10.5　计算机病毒的分类 …… 426
10.5.1　传统的计算机病毒分类 …… 426
10.5.2　恶意程序、宏病毒、脚本病毒、蠕虫、特洛伊木马 …… 431
10.6　计算机病毒的传播 …… 437
10.7　计算机病毒的结构和工作机理 …… 438
10.7.1　病毒的引导机理 …… 440
10.7.2　病毒的传染机理 …… 441
10.7.3　病毒的破坏（表现）机理 …… 441
10.7.4　病毒的触发机理 …… 442
10.8　计算机病毒的检测 …… 443
10.8.1　特征代码法 …… 444
10.8.2　校验和法 …… 445
10.8.3　行为监测法 …… 446
10.8.4　软件模拟法 …… 446
10.8.5　比较法 …… 447
10.8.6　感染实验法 …… 447
10.8.7　分析法 …… 448
10.9　计算机病毒的清除 …… 448
10.9.1　文件型病毒的清除 …… 449
10.9.2　引导型病毒的清除 …… 453
10.10　计算机病毒的预防 …… 455
10.10.1　引导型病毒的预防 …… 455
10.10.2　文件型病毒的预防 …… 456
10.10.3　宏病毒的预防 …… 456
10.10.4　电子邮件病毒的预防 …… 457
10.11　本章小结 …… 458
习题 10 …… 459
参考文献 …… 461



第一部分

基　础　篇

第 1 章
信息系统安全绪论



兵法曰：知彼知己，百战不殆。攻与防的对抗是信息安全的主题，了解安全攻击才能更好地进行安全防御。本章以信息安全攻击的实景为起点，初步建立了信息安全攻击威胁的感性认识，开始体验信息安全的基本理念，并透过信息安全的攻击威胁感受系统安全在信息安全中的地位和作用。本章主要由安全攻击实景呈现、安全攻击环节概览、信息安全典型事件、信息安全经典要素、信息系统安全策略、信息系统访问控制及系统安全知识定位等内容构成。

1.1　安全攻击实景呈现

研究信息安全问题的目的是解决在现实应用中遇到的日益棘手的信息安全问题。本着“从实践中来，到实践中去”的思想，作为本章乃至本书的开始，下面讲述一个并非虚构的故事。

故事发生在一个发达国家，故事的主人公名叫卡尔，深谙信息安全攻击之法，他可不是一个喜欢通过恶作剧来炫耀自己的人，他实施信息安全攻击的意图非常明确，那就是获取经济利益。

1.1.1　诱惑及初探

一个偶然的机会，卡尔在媒体上看到了一篇有关某公司发展情况介绍的文章，那是一家销售精致小饰品的公司，据文章介绍，该公司发展非常迅速，一时间销售网点已遍布全国各地。

读了该介绍文章，卡尔顿时眼前一亮，似乎看到了商机。他想：这家公司扩张得如此迅速，想必在网络信息系统建设中也许还来不及慎重考虑安全防范的问题，在信息安全方面应该是有机可乘的。于是，卡尔盯上了该公司，从现在开始，我们不妨把该公司称为猎物公司，因为卡尔已经把它纳入了自己的攻击对象。

在发动攻击之前，卡尔需要收集有关猎物公司的更多资料，他立刻开始了针对猎物公司的侦察行动。卡尔首先想到的就是光顾提供互联网域名登记服务公共信息的互联网信息中心 InterNIC 的网站，尝试从中收集猎物公司的有用信息。这一试还颇有收获，卡尔从中得知猎物公司的 IP 地址空间范围是 *a.b.c.*0～255。

卡尔决定借助猎物公司的 IP 地址空间信息对该公司的网络信息系统进行扫描。为了防止入侵检测系统（IDS）的检测及入侵防御系统（IPS）的阻拦，卡尔在互联网中找到一个可利用的系统，在该系统中安装上 FragRouter 软件，以该系统作为中转进行扫描，避免自己的系统直接暴露在扫描信息通信的前方，通过第三方进行扫描的方案如图 1.1 所示。

卡尔使用 Cheops-ng 软件工具探测猎物公司的网络系统，查看猎物公司网络中有什么样的系统在运行。扫描发现，通过互联网可以访问到猎物公司网络中的三个系统。利

用集成在 Cheops-ng 中的 traceroute 功能，卡尔获知，从系统布局上看，在这三个系统中，有一个系统位于其他两个系统的前端。

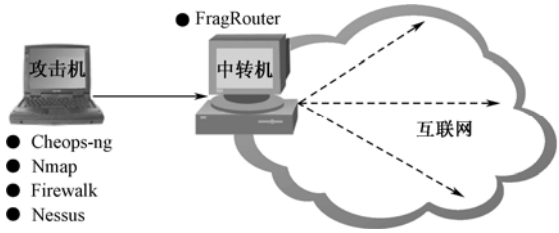


图 1.1　通过第三方系统进行扫描的方案

使用 Nmap 软件工具对这几个系统进行 SYN 扫描，卡尔发现其中有一个系统的 TCP 80 端口是打开的，这表明该系统是一个 Web 服务器。另外两个系统没有打开的 TCP 端口，但用 Nmap 软件工具进行 UDP 扫描发现有一个系统的 UDP 53 端口是打开的，这表明该系统是一个 DNS 服务器。另外一个系统没有任何打开的端口，卡尔用 Firewalk 软件工具扫描发现该系统是一个包过滤防火墙，它的规则允许通过 TCP 80 端口和 UDP 53 端口对猎物网络的非军事区（DMZ）进行访问。至此，卡尔基本摸清了猎物公司网络的互联网 DMZ 和防火墙的一般结构，如图 1.2 所示。

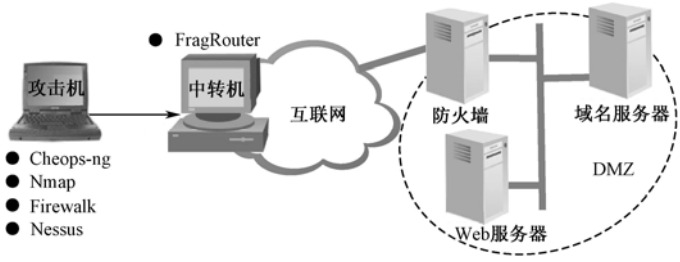


图 1.2　猎物公司网络的互联网 DMZ 和防火墙的基本结构

卡尔运行 Nessus 软件工具对猎物公司网络的 DMZ 进行漏洞扫描，查看其中有没有可以利用的简单漏洞存在。例如，检查一下其中是否存在有安全漏洞的服务或没有打安全漏洞补丁的服务可以利用。不幸的是，Nessus 漏洞扫描并没有为卡尔发现猎物公司网络的 DMZ 中有可以利用的漏洞。

1.1.2　确定合适的突破口

首战未果并没有打消卡尔攻击猎物公司的念头，他静下心来，浏览猎物公司的网站信息，希望从中获得采取下一步行动的启发。猎物公司网站的一个网页描述了该公司销售网点的分布情况。卡尔很快就发现，在离他家不远的地方，就有猎物公司的一个销售点，他想：真是天赐良机。

于是，带上自己的笔记本电脑，卡尔驱车来到了该销售点附近。就在自己的小汽车

里，卡尔启动了笔记本电脑，运行 Wellenreiter 软件工具，试图探测猎物公司销售点的无线网络接入点。Wellenreiter 是一款无线网络检测工具，它可以探测出无线网络的接入点，即使接入点配置为使通信包不包含表示网络名的 SSID 信息或设置为不响应探测包。通过查看 Wellenreiter 的合法通信信息，卡尔可以看到附近有若干无线网络接入点，其中一个 SSID 为 acwicorp041 的接入点引起了他的注意，他断定：那一定是猎物公司销售点的无线网络接入点，因为该公司的英文名称是“Acme Widgets Corporation”，而该 SSID 正隐含着该英文名称的缩写。

卡尔以 acwicorp041 为 SSID 配置他的笔记本电脑上的无线客户端，试图连接相应的无线接入点。可是，连接并不顺利。该接入点好像具有包过滤功能，扔掉了他发送的所有通信包，不响应他的接入请求。卡尔仔细分析 Wellenreiter 的显示信息，忽然注意到了使用 acwicorp041 这个 SSID 的其他设备的 MAC 地址，这使他立刻想到：该接入点可能采取了 MAC 地址绑定措施，只允许具有指定的 MAC 地址的设备进行连接。

卡尔从 Wellenreiter 的显示信息中选取一个 MAC 地址，使用他的笔记本电脑的 Linux 系统中的 ifconfig 命令把自己的 MAC 地址伪造成该 MAC 地址，再次试着进行连接。果然连接成功，他的笔记本电脑系统连接到了猎物公司网络的销售点 A 的内部网络中，并由该网络根据 DHCP 分配了自动的 IP 地址。

连接上了销售点 A 的内部网络系统后，卡尔使用 Nmap 工具中的 ping 扫描功能对该网络进行扫描，发现了连接到该网络中的其他无线 POS 机设备和销售点 A 的服务器的 IP 地址，使用 Linux 系统中的 dig 命令和该服务器的 IP 地址进行逆向 DNS 查询，他确定出该服务器的域名是 store041.internal_acmewidgets.com。

根据 Nmap 的扫描结果，卡尔还发现，销售点 A 的服务器打开了 TCP 5900 端口，这表明该服务器可能运行着虚拟网络控制（VNC）服务。VNC 是一个常用的基于 GUI 的工具，允许管理员通过网络对系统进行远程控制，具有系统控制功能，通过它可以获得系统的重要信息。卡尔运行 THC Hydra 口令猜测工具对销售点 A 的 VNC 服务的口令进行猜测，该工具对 root、admin 和 operator 等一系列标准的用户账户进行逐个口令的猜测。令卡尔高兴的是，该工具发现 operator 账户的口令是 rotarepo，这仅是账户名字符串的逆向表示。

掌握了销售点 A 的 VNC 账户和口令信息，卡尔便可以通过 VNC 堂而皇之地登录进入销售点 A 的服务器，并具有系统控制权。在这样的情况下，卡尔可以随意地对系统中的文件进行浏览，寻找他感兴趣的信息。

终于，在销售点 A 的服务器中的一个命名古怪的目录下，卡尔发现了一个有价值的文件，该文件记录了 POS 终端机存放到销售点 A 的服务器上的交易历史信息，共提供了该销售点 100 天以上的交易记录，其中包含了所有的信用卡信息（如卡号、持卡人姓名和有效日期等）。实际上，自该销售点服务器部署以来的所有交易信息都在该文件中。

结果，仅从销售点 A 的服务器上，卡尔就获得了超过 10 万张信用卡的信息。至此，卡尔对猎物公司销售点 A 的攻击宣告成功。

获取信用卡信息正是卡尔对网络信息系统实施安全攻击所追求的目标，他总是以每张信用卡 1 美元的价格在黑市上出售信息卡信息，以此获利。当然，对于卡尔来说，每张 1 美元的价格是微不足道的，但他的策略是薄利多销，以数量取胜。

1.1.3　设法扩大战果

因为卡尔是要靠数量发财的，所以，10 万张信用卡的信息不足以令他感到满足，况且，多方面的资料显示，猎物公司拥有遍布全国各地的 200 多个销售点，仅仅得手一个销售点岂能让他善罢甘休。而且，俗话说，万事开头难，现在，卡尔已经成功地攻入了猎物公司销售点 A 的服务器系统，这意味着在攻击猎物公司的网络信息系统的道路上，他已经打开了一个好的局面，也许，下一步的工作就容易得多了。

无论从哪个方面说，卡尔都不会就此终止对猎物公司的兴趣。他回顾了对销售点 A 的攻击过程，自然会想到要攻击其他的销售点，但是，依次驱车前往各个销售点显然是件费事费时的差事。

分析成功攻入销售点 A 的服务器系统的基本要点，关键是该服务器系统提供了 VNC 服务并选择了容易猜测的口令。考虑到猎物公司发展速度之惊人，卡尔推测各销售点的网络信息系统可能采用相同的模式进行部署，因而可以采取类似的方法尝试攻击。

第二天，卡尔照样驱车来到销售点 A，希望借助在该销售点的无线接入发动对其他销售点的攻击。他依然想采取 VNC 登录的方式去控制系统，但这一次，他不是采用与上一次相同的 IP 地址 *w.x.y.z* 去连接销售点 A 的 VNC 服务器，而是把 IP 地址调整为 *w.x.y*+1.*z*，居然也连接成功了，连接到了另一个 VNC 服务器，甚至这个 VNC 服务器上的账户名和口令也与销售点 A 的完全一样。可见，猎物公司是采用完全照搬的方式部署各个销售点的网络信息系统的。这一次，卡尔真可谓不费吹灰之力便拿下了销售点 B，又得到了另一批信用卡的信息。

如此这般，照说卡尔应该可以采取该方法逐个攻下各个销售点了。但要彻底把所有的销售点都过一遍，至少也得重复类似的工作 200 多次，这样做也并不省事。因此，卡尔设法尝试更加便捷的方法。

1.1.4　全面出击

卡尔再度分析对销售点 A 的服务器进行端口扫描得到的结果，他发现，该服务器运行着一个常用的备份程序，而且，该备份程序含有缓冲区溢出漏洞。

凭借攻击销售点 B 的 VNC 服务获得成功的直觉，卡尔推断销售点 B 的服务器也同样运行着相应的备份程序，因此，他在他的笔记本电脑的 Linux 操作系统中运行

Metasploit 攻击工具，对销售点 B 的服务器上的备份程序发动缓冲区溢出攻击，不出所料，他果然获得了成功。

这时，在 Metasploit 的支持下，卡尔获得了销售点 B 的服务器的命令解释程序的完全控制权，借此机会，他在该服务器上安装了一个嗅探程序，以便捕获通过销售点 B 的网络系统传输的各种信息。

销售点 B 的服务器中的嗅探程序为卡尔提供了该销售点上的 POS 终端机传给服务器的交易信息，诚然，这些信息对卡尔来说并没有太大的意义，因为通过该销售点上的 VNC 服务，他已经能够得到这些信息。不过，该嗅探程序还提供了非常有价值的信息，它显示，销售点 B 的服务器还向其他网络中的其他服务器发送交易请求，而且，这些交易请求是以明文传送的，经过分析，卡尔发现，这些正是信用卡授权请求信息。

利用嗅探程序提供的接收授权请求信息的目的服务器的地址，卡尔运行 Nmap 工具对该目的服务器进行端口扫描。这一次，卡尔扫描的实际上是猎物公司总部的中央网络系统中的服务器，它负责处理所有的信用卡交易并管理公司的业务。Nmap 扫描结果显示，该服务器打开了 TCP 443 端口，这表明该服务器应该是提供了一个 HTTPS 服务。

通过笔记本电脑的 Linux 操作系统上的浏览器，卡尔对猎物公司总部服务器支持的给定网站进行浏览，只看到一个描述公司内部管理应用系统的页面，该页面并没有提供敏感信息，它只允许猎物公司的内部员工登录后进入一个提供详细的业务信息的 Web 应用程序。没有合法的账户和口令是无法进入该应用系统的。卡尔尝试使用与 VNC 服务相同的账户和口令，但并不奏效，使用 THC Hydra 工具进行口令猜测也没有取得成功。

看来，卡尔必须另寻他径了。此刻，卡尔启动了 Paros Proxy 工具，利用该工具的 Web 应用自动扫描功能，卡尔在该猎物公司的 Web 应用程序中查找跨站点的脚本和 SQL 注入缺陷。卡尔感到很庆幸，该工具在与目标应用系统的用户账户相关联的 Web cookie 中发现了 SQL 注入缺陷。

通过在 Paros Proxy 工具中设置手工操作 cookie 的工作模式，卡尔利用 SQL 注入缺陷，获得了对猎物公司的目标 Web 应用系统的底层数据库进行访问的机会。在该数据库中，卡尔发现了存放猎物公司全部 200 多个销售点的所有顾客信息的一张表，其中含有 100 多万张信用卡的信息，这正是卡尔苦苦寻找的宝藏，他终于大功告成了。

因为攻击的预定目的已经达到，所以，卡尔随即撤离了现场，返回家中。接下来，通过他的地下销售途径，迅速出售所有信用卡信息。最后，他销毁了所有与这次信用卡信息攻击行为有关的信息，开始度假，享受胜利果实。

1.1.5　尾声

卡尔的安全攻击措施是有效的，但有关机构的安全响应措施也并非一无是处。两个星期之后，信用卡公司发现：一时间，出现了大量的信用卡欺骗行为。

通过对欺骗行为的相关系统审计信息进行分析，信用卡公司发现，涉案信用卡在过去的几个月里都曾经在猎物公司的销售点进行过交易，由此意识到可能是猎物公司出现了问题，并立即通知了猎物公司。

猎物公司由此展开了内部调查，并证实了安全事件的发生。按照有关法律规定，猎物公司不得不告知所有信息被盗的信用卡持卡人，并履行相应的赔偿。毫无疑问，在由卡尔发动的这次信用卡信息盗窃安全事件中，猎物公司遭受了惨痛的声誉损失和经济损失。

1.2　安全攻击环节概览

结合 1.1 节讲述的故事，可以概要地了解一下在攻击者实施的安全攻击过程中所涉及的主要环节。一般而言，网络信息安全攻击可以划分为侦察、扫描、获取访问、维持访问和掩盖踪迹等阶段。下面分别对各个阶段的基本思想、技术、工具和对策进行简要的介绍。

1.2.1　侦察

很多安全攻击都始于侦察阶段，在该阶段，攻击者想方设法收集尽量多的有关被攻击目标的信息，以便为实际的攻击做好准备。

在侦察阶段，攻击者广泛采用的手段的技术含量通常并不高。社会工程学便是常见的手段之一，它的特点就是通过普通谈话来诱导受害者透露敏感信息。电话交谈是社会工程学的常见形式，攻击者在电话中以花言巧语从用户口中套出电话号码、姓名、口令或其他敏感信息。攻击者也可能伪造来电号码以使受害者更容易上当。

淘垃圾也是一种值得关注的侦察手段，指的是从一个单位遗弃的办公用品中搜寻敏感信息的过程，淘垃圾者可以从被扔掉的粘贴便条、光盘、旧硬盘等中获得系统资料、用户名单、电话号码甚至口令信息。

单位的 Web 网站也是攻击者在侦察中可以充分利用的重要资源，很多单位都把员工联系信息、合作伙伴信息及应用技术信息等挂到了 Web 网站上，这些信息对于攻击者来说可能具有重要的价值，而且只须使用诸如 Google 的搜索引擎就可以获得。

从 Whois 数据库中可以找到攻击目标的互联网地址、域名和通信联系等方面的信息，InterNIC 为.com、.net、.org 及其他顶级域名提供 Whois 数据库。Uwhois Web 网站可用于研究在 246 个不同国家登记的各种域名。

从 DNS 服务器中也可以获得很多有用的信息，其中包括域名到 IP 地址的映射、邮件服务器列表及其他名字服务器等。DNS 是一个用于解析域名的分布式的层级结构的数据库。

Windows 操作系统和 UNIX 操作系统中的 nslookup 工具可用于访问 DNS 服务器，

从 DNS 服务器中获取给定域名的相关信息。

互联网上有很多通用的侦察工具可供下载使用，其中，最有用的一个就是 Sam Spade，它提供了很多获取侦察信息的方法。另外，也有大量基于 Web 的工具可用于侦察。

1.2.2 扫描

完成了侦察阶段的信息收集工作之后，攻击者通常要对系统进行扫描，以便进一步掌握有关攻击目标的更多信息。扫描阶段的常见工作包括无线接入扫描、拨号扫描、网络映射、端口扫描、防火墙规则扫描及漏洞扫描等。

无线接入扫描是寻找目标网络的无线接入点并确定它们的 SSID 的过程，SSID 可以表示网络的名称。Windows 操作系统中最流行的无线接入扫描工具是 NetStumbler，它是一个主动扫描工具，它通过发送 SSID 域的值为 any 的探测包去检测无线网络，大部分无线接入点都会响应它发出的探测请求。Wellenreiter 是一个被动无线接入扫描工具，它通过嗅探无线通信频率去检测无线局域网（WLAN）。防御无线接入扫描的方法之一是采用较强的无线认证协议，如 WPA 或 802.11i。

拨号扫描是以逐个拨打电话号码的方式去检测调制解调器载波音频进而寻找不安全的调制解调器的过程。不安全的调制解调器是进入目标网络的最容易的渠道之一。攻击者可以从 Web 网站中查找相应的电话号码号段，继而用来进行拨号扫描。发现调制解调器后，再寻找不设口令的系统或口令容易猜测的系统。THC-Scan 是最流行的拨号扫描工具之一。

网络映射是确定目标网络中的机器清单及这些机器所构成的网络拓扑结构的过程。攻击者首先对目标网段进行扫描，确定该网段中有哪些主机在工作，然后利用 traceroute 功能确定系统、路由器和防火墙是如何连接在一起的。Cheops-ng 是一个可用于执行网段扫描和 traceroute 功能的非常有用的工具。

端口扫描是确定目标网络中哪些端口正在监听服务请求的过程。端口扫描程序通过与目标系统中的各种端口进行对话，可以确定系统中所有正在运行的服务的清单。功能最丰富的端口扫描工具之一是 Nmap，它支持进行多种类型的扫描，包括 TCP SYN 扫描、TCP ACK 扫描、UDP 扫描等。Nmap 还包含操作系统指纹提取功能，可以确定目标机器上的操作系统类型。

防火墙规则扫描是确定目标网络中的包过滤防火墙所实现的规则的过程，Firewalk 是一个可用于实施这样的扫描的有用工具。

漏洞扫描是检测目标系统中可能存在的安全漏洞的过程。漏洞检测工具可用于对目标系统中成百上千的安全漏洞进行检测，它们借助记录已知的配置错误、系统 Bug 和其他问题的数据库进行工作。Nessus 是漏洞扫描的最好工具之一。

实施扫描时，攻击者常常采用各种措施避开目标环境中部署的 IDS 和 IPS 的防范，FragRouter 和 FragRoute 工具常用于避开网络层的 IDS 和 IPS 防范，Nikto 工具常用于避开应用层的 IDS 和 IPS 防范。

1.2.3　获取访问

利用侦察和扫描阶段收集到的信息，攻击者可以尝试闯入目标系统、对目标系统进行访问。攻击者可能采取对操作系统、应用程序或网络系统进行攻击的方法获取对系统的访问权。与此相关，攻击者也可能发动拒绝服务攻击，破坏系统为合法用户提供的正常访问。

缓冲区溢出攻击是最常用和最具破坏性的攻击方法之一，它允许攻击者通过程序输入把可执行代码输入到目标系统中执行，为攻击者在目标系统中执行恶意代码提供了可能。缓冲区溢出呈现为栈溢出和堆溢出两种类型。Metasploit 是攻击者常用的一个集成化的缓冲区溢出攻击工具。

口令攻击也是攻击者为闯入系统而发动的常见攻击方式之一，常见的口令攻击方法包括字典攻击、暴力攻击或混合攻击。Cain 是 Windows 系统中破解口令的最好工具之一，而 John the Ripper 工具则在 UNIX 系统中表现出色。

基于 Web 的应用的流行和普及也使它们成为攻击者为了闯入系统而进行攻击的重要目标， 常见的攻击方法包括账户捕获、会话捕获和 SQL 注入等。账户捕获就是根据应用程序响应不同的账户输入时返回的不同错误信息来确定登录系统用的账户名甚至口令，从而登录进入应用程序。会话捕获就是根据应用程序的 URL 参数、隐藏的表格元素及 Cookie 等来克隆合法用户与应用程序间的会话号，从而获得对应用系统的访问。SQL 注入就是在向应用程序提供的用户输入中添加 SQL 元素，从而扩展应用程序中的 SQL 语句，获得对应用程序的后端数据库的非法访问。

Web 浏览器中的各种漏洞也常被攻击者利用，当用户运行有漏洞的浏览器访问受感染的 Web 服务器时，攻击者就有可能控制用户的机器。通过攻陷可信的 Web 服务器，攻击者就能在浏览这些服务器的用户不知不觉的情况下控制他们的机器。

嗅探、IP 地址欺骗和会话劫持等是网络攻击的常见手段。嗅探就是在局域网（LAN）中收集网络通信信息，这些信息可能包含明文传送的用户 ID 和口令、敏感文件或电子邮件等。嗅探可分为被动嗅探和主动嗅探。被动嗅探只从网上采集信息，不操纵网上的信息流，被动嗅探的常见工具有 Snort、Sniffit 及 Ethereal 等。主动嗅探则向网中注入通信信息，主动把网络通信包改向传递到嗅探机器中，Dsniff 和 Ettercap 是两个功能强大的主动嗅探工具。

IP 地址欺骗就是攻击者伪造他所发送的信息的源 IP 地址，冒充其他信息发送方，欺骗信息的接收方。这种攻击方法常用于设计圈套、逃避过滤或接入基于 IP 地址认证

的系统。

会话劫持就是综合运用嗅探和 IP 地址欺骗手段，把合法用户已经建立的合法会话据为己有，从而获得对系统的访问。会话劫持攻击可能发生在网络中，也可能发生在单个主机系统中。

拒绝服务攻击的基本方法是使系统服务停止工作或把系统资源消耗殆尽，这两种方法既可以在系统本地实施，也可以通过网络远程实施，从而形成四种主要的拒绝服务攻击类型。

直接杀死提供服务的进程、通过系统配置关闭指定的服务或促使服务崩溃等是常见的本地停止服务型攻击，逻辑炸弹属于这种攻击类型。填满进程分配表、占满文件系统空间或用尽网络通信连接等是常见的本地资源耗尽型攻击。发送不良网络包，利用目标操作系统或应用程序中的 Bug，促使系统崩溃，是典型的远程停止服务型攻击。在网络中产生大量的通信包则可以发动远程资源耗尽型攻击，SYN 洪水攻击是属于这种类型的常见攻击。

1.2.4　维持访问

攻入一个系统终归不是一件轻而易举的事情，一旦攻入成功，攻击者总要采取各种可能的措施，以便为下一次攻击打开方便之门，这就是维持访问所要实现的目标，木马和后门是实现该目标的主要手段。

木马是貌似非常有用却暗藏恶意功能的程序。后门是使攻击者能够避开正常的安全控制而得以访问系统的程序。木马与后门融合而成的木马后门则是最具破坏性的维持访问攻击程序。

根据工作空间所处的系统层次，木马后门可划分为应用层木马后门、用户态 rootkit 和内核态 rootkit 三种类型。

应用层木马后门是可以运行在目标机器上的独立应用程序，表面上看不会产生任何安全问题，但实际上能够使攻击者访问并控制目标机器。远程控制程序、僵尸程序（Bot）和间谍软件是三类最为流行的应用层木马后门，攻击者可以借助它们在目标机器上访问任何文件、观察用户行为、记录键盘输入及实施其他操作。

用户态 rootkit 是植入到用户态的操作系统关键程序中的木马后门，在 UNIX 和 Linux 操作系统中，攻击者常用用户态 rootkit 替换诸如 login 和 sshd 等系统程序，为访问系统留下后门，并在该基础上，进一步替换诸如 ifconfig、ls、ps 和 du 等系统程序，以便使他们访问系统的行为能够躲开系统管理员的视线。使用文件系统完整性检查工具 Tripwire 有助于发现系统中的用户态 rootkit。

内核态 rootkit 则是植入到操作系统内核中的木马后门。大部分的内核态 rootkit 都提供执行重定向功能，该功能可以把用户的请求重新映射为执行攻击者指定的程序。内核

态 rootkit 还支持对文件、目录、TCP 和 UDP 端口及运行中的进程进行隐藏。为了防止内核态 rootkit 攻击，应该及时给操作系统打补丁，启用基于主机的 IPS，以便能够在第一时间防止攻击者获得对系统进行特权访问的机会。

UNIX 和 Linux 操作系统中的 Chkrootkit 和 Rootkit Hunter 工具及 Windows 操作系统中的 Rootkit Revealer 和 Blacklight 工具可用于检测用户态 rootkit 和内核态 rootkit 给系统带来的异常。

1.2.5　掩盖踪迹

为了既能够成功地对网络信息系统实施安全攻击、实现攻击目标，又可以逍遥法外，攻击者总是要设法隐藏他们的攻击行为，以期不被发现。特别是在安装了维持访问的木马后门之后，攻击者总要操纵系统，以便销毁攻击痕迹、掩盖攻击行径。

销毁攻击痕迹的最重要的途径之一是修改系统的日志记录信息。完全删除日志信息并不明智，因为这样很容易被系统管理员发现，所以，攻击者一般不会这样做，一般只修改与某些事件相关的日志信息。攻击者通常修改反映他们攻击系统的行为的日志记录。例如，失败的登录尝试、特殊账户的使用及安全敏感命令的运行等。

在 Windows 操作系统中，可以使用 WinZapper 工具删除特定安全事件的信息。在 UNIX 操作系统中，有很多工具可用于修改日志记录信息。

为了防止日志记录信息被修改，可以采用独立的日志服务器来记录关键网络（如 Internet 中的 DMZ）的日志信息。另外，也可以对日志文件进行加密，使得攻击者就算控制了系统也无法修改日志信息。还可以定期地把日志信息转存到只能单次写的介质上（如 DVD 等），以便为日志记录留下攻击者无法修改的永久备份。

1.3　信息安全典型事件

随着计算机和互联网技术的迅猛发展，其应用领域正在不断拓展，在电子商务、公用网站、金融电子化等领域的广泛深入极大地改变了人们传统的生活和学习方式。技术的普及使得越来越多的人开始接触并使用计算机，但是随着社会网络化程度的增加，开放式网络空间中的信息安全隐患日益明显地暴露出来。

1988 年的莫里斯蠕虫事件揭开了大规模网络攻击的序幕，虽然该程序的攻击方法很简单，但是它所造成的损失至今也难以清楚地估计出来。

自此，网上不甘寂寞的好事者纷纷效仿，制造了一系列骇人听闻的“黑客事件”。下面就是其中一些典型的信息安全事件。

1995 年 2 月，被认为是世界上头号黑客的凯文 • 米特尼克（Kevin Mitnick）再次被送上了法庭。联邦调查局从 1992 年就开始搜捕他，但直到 1995 年才在北卡罗来纳州的罗利将其抓获。因其在摩托罗拉、诺基亚、北电、太阳微系统等知名公司和南加

州大学的网站上窃取软件并篡改数据，由此造成了数千万美元的损失而被判监禁 5 年。

1996 年初，法国国防部证实：法国海军参谋部计算机储备的军事机密于 1995 年 7 月被人窃走，其中包括几百艘盟军军舰的声音识别密码及舰只航行图。

1997 年 11 月，工行青岛分行信用卡部一名工作人员利用工作之便解析信用卡密钥生成技术，制造假牡丹卡诈骗 70 多万元。在我国，仅 1997、1998 两年，四家国有商业银行就发生计算机犯罪案件共 140 余起，涉及人员 160 多人，涉案金额达 1.6 亿元，造成重大经济损失。

1999 年 4 月 26 日，全世界至少有 6000 万台计算机遭受了 CIH 病毒的侵害，计算机系统瘫痪或硬盘分区表被改写，甚至许多机器的数据不可逆转地永久丢失。这以后每到 4 月 26 日，甚至每月的 26 日，计算机用户都心有余悸。梅莉莎病毒、蠕虫病毒、“我爱你”病毒都成为每年计算机安全事件的热门。

2000 年 2 月，以“雅虎”为首的美国一系列大型网站遭到了黑客有组织的攻击，他们攻击的目标包括了雅虎、电子港湾、亚马逊、微软网络等美国大型国际互联网网站。据统计，在 7、8、9 日这短短的三天里，这些受害公司的损失就超过了 10 亿美元，其中仅营销和广告收入一项便高达 1 亿美元。4 月份，威尔斯少年黑客葛雷在窃取信用卡资料时，也顺便看了看微软总裁盖茨的资料，10 月中旬，微软的高度机密网络被一来自俄罗斯的黑客入侵，在几周的时间里，黑客攻破了微软的系统，并偷走了微软的部分源代码，引起全球恐慌。

2001 年 1 月，首都在线 263 的一个服务器受到黑客的攻击。该网站的页面被修改，而且黑客在网页上还“签上”了自己的名字。4 月 4 日，美国一些黑客组织相继对我国的一些政府、企业、教育、科研、电信等网站进行攻击。特别是中美撞机事件发生以来，我国的一些黑客组织发起了一场网络反击战。这样，便引发了 IT 发展史上一个十分具有历史意义的事件——中美黑客大战。2001 年 8 月 12 日，中国一些黑客组织在获知日本首相小泉参拜靖国神社后，立即采取行动，替换了部分日本政府的主页，以示抗议，有 10 余个日本政府的网站被黑。

2002 年 3 月，有人闯入了 SalesGate.com 的 B2B（Business-to-Business）网站，窃取了约 3000 个客户记录，其中包括信用卡卡号及其他个人信息。此人还在互联网上张贴了其中一些人的资料。

据美国金融时报报道，世界上平均每 20s 就发生一次入侵国际互联网的计算机安全事件，三分之一的防火墙被攻破。

美国联邦调查局计算机犯罪组负责人吉姆·塞特尔称：给我精选 10 名黑客，组成一个小组，90 天内，我将使美国趴下。

美国国防部对其军用计算机网络进行安全检测后所提出的报告指出：现有网络中，85%的计算机可能受到侵害；15%曾有报警记录；5%曾有遭受过攻击的报告。

美国生产出了第一代采用“病毒固化”技术的芯片，并开始嵌入出口的计算机产品中。一旦需要，便可遥控激活。在海湾战争中，美国特工人员在安曼将伊拉克从德国进口的一批计算机打印设备换上含有“可控计算机病毒”的芯片，导致伊方的计算机系统在战争初期就陷入全面瘫痪。

据总部设在卡内基•梅隆大学软件工程学院的美国计算机紧急事件反应小组（CERT）的统计，1988 年发生的计算机安全事件仅 6 起，其中包括被认为是大规模互联网攻击的开端的“莫里斯蠕虫事件”。到 2003 年年底，计算机安全事件达到 137 529 起，自 1988 年统计以来到 2003 年，共计发生 319 992 起计算机安全事件。由 CERT 小组统计的 1988—2003 年各年发生的计算机安全事件数如表 1-1 所示。

表 1-1　1998—2003 年计算机安全事件数

年　份	2003	2002	2001	2000	1999	1998	1997	1996	1995	1994	1993	1992	1991	1990	1989	1988
事件数	137 529	82 094	52 658	21 756	9 859	3 734	2 134	2 573	2 412	2 340	1 334	773	406	252	132	6

据初步统计，仅在美国，黑客事件每年造成的经济损失就超过 100 亿美元。涉及政府机构、军事部门、科研院校、金融商业等部门的计算机犯罪，严重干扰了人们的日常生活，恶意侵犯公民隐私，造成巨大的经济损失，甚至直接或间接地威胁到国家安全。严峻的现实让人们清醒地意识到计算机和网络的发展离不开信息安全技术的保障。随着人们安全意识的提高，安全领域的探索和研究日益深入。世界各国都投入了大量的人力、物力和财力，不遗余力地提高计算机信息系统的安全性。

1.4　信息安全经典要素

机密性、完整性和可用性是信息安全的三个经典要素，这三个经典要素对应着信息安全的三种基本的安全服务。对这三种安全服务的解释随着它们所处环境的不同而不同。在某种特定环境下，对其中某种安全服务的解释也是由个体需求、习惯和特定组织的法律所决定的。

1.4.1　机密性

机密性是指对信息或资源的隐藏。信息保密的需求源自计算机在敏感领域的使用，如政府或企业。政府中的军事、民事机构经常对需要获得信息的访问人群设定限制。由于军事部门试图实现控制机制以体现“需要知道（need to know）”原则，这就导致了计算机安全领域中最早的形式化研究工作的出现。这一原则同时也适用于产业公司，它可保护公司的专利设计的安全，避免竞争者窃取设计成果。进一步的例子是：所有不同类型的机构中都会对机构中的人事记录进行保密处理。

访问控制机制支持机密性。其中，密码技术就是一种保护机密性的访问控制机制，这种技术通过编码数据，使数据内容变得难以理解。密钥控制着非编码数据的访问权，然而这样使得密钥本身又成为另一个有待保护的数据对象。

例 1.1　加密一份退税单可防止任何人阅读它。如果税务申报人需要查看它，必须通过解密。只有密钥的拥有者才能够将密钥输入解密程序。然而，如果在密钥输入解密程序时，其他人能读取该密钥，则这份退税单的机密性就将被破坏。

其他一些依赖于系统的机制能够防止信息的非法访问。不同于加密数据的是：受这些机制保护的数据只有在机制失效或机制被旁路时才可以被读取，因此它们的优点中也存在相应的缺点。这些机制能比密码技术更完全地保密数据，但如果机制失效或被攻破，则数据就变为可读的了。

机密性也同样适用于保护数据的存在性，存在性有时比数据本身更能暴露信息。精确地知道有多少人不信任某个政客，可能还不如知道该政客的参谋人员曾经举办过这样一次民意测验更重要；准确地知道某个特定的政府情报局是如何骚扰其国民的，也可能不会比知道这样的骚扰曾经发生过更重要。访问控制机制有时仅隐藏数据的存在性，以免存在性本身暴露了应该受到保护的信息。

资源隐藏是机密性的另一个重要课题。某个站点通常希望隐藏其配置信息及其正在使用的系统。机构也不希望别人知道他们的某些特有设备（否则这些设备会被非法使用或被误用）；同样，从服务提供商处获取计时租借服务的公司不希望别人知道他们正在使用哪些资源。访问控制机制能提供这些功能。

所有实施机密性的机制都需要来自系统的支持服务。其前提条件是：安全服务可以依赖于内核或其他代理服务来提供正确的数据，因此，假设和信任就成为保密机制的基础。

1.4.2　完整性

完整性指的是数据或资源的可信度，通常使用“防止非法的或未经授权的数据改变”来表达完整性。完整性包括数据完整性（即信息的内容）和来源完整性（即数据的来源，常通过认证来确保）。信息来源可能会涉及来源的准确性和可信性，也涉及人们对此信息所赋予的信任性。这种二分法显示了这样的原则：完整性中的可信性对系统的固有功能起到重要作用。

例 1.2　某份报纸可能会刊登从白宫泄露出来的信息，却声称信息来自于另一个信息源。信息按原样刊登（即保持数据完整性），但是信息的来源不正确（即破坏了来源完整性）。

完整性机制可分为两大类：预防机制和检测机制。

预防机制通过阻止任何未经授权的改写数据的企图或任何使用未经授权的方法来改写数据的企图，来确保数据的完整性。区分这两类企图是重要的，前者发生在当用户企图改写其未经授权修改的数据时，而后者发生在已授权对数据做特定修改的用户试图使用其他手段来修改数据时。例如，假设有人入侵一个计算机财务系统，并试图修改账目上的数据，即一个未经授权的用户试图破坏财务数据库的完整性。但如果是被公司雇用来维护账簿的会计想挪用公司的钱款，他把钱款寄到海外，并隐瞒这次交易过程，即用户（会计员）企图用未经授权的方法（把钱款转移到海外账户上）来改变数据（财务数据）。适当的认证与访问控制一般都能够阻止外部入侵，但要阻止第二类企图就需要一些截然不同的控制方法。

检测机制并不试图阻止对完整性的破坏，它只是报告数据的完整性已不再可信。检测机制可以通过分析系统事件（用户或系统的行为）来检测问题，或者（更常见的是）通过分析数据本身来查看系统所要求或期望的约束条件是否依然满足。检测机制可以报告数据完整性遭到破坏的实际原因（某个文件的特定部分被修改），或者仅报告文件现在已被破坏。

处理机密性与处理完整性有很大差别。对机密性而言，数据或者遭到破坏，或者没有遭到破坏；但是完整性则要同时包括数据的正确性和可信性。数据的来源（即如何获取数据及从何处获取数据）、数据在到达当前机器前所受到的保护程度、数据在当前机器中所受到的保护程度等都将影响数据的完整性，因此，评价完整性通常是很困难的，因为这依赖于关于数据源的假设及关于数据源的可信性的假设，这是两个常常被忽视的安全基础。

1.4.3　可用性

可用性是指对期望的信息或资源的使用能力。可用性是系统可靠性与系统设计中的一个重要方面，因为一个不可用的系统所发挥的作用还不如没有系统。可用性之所以与安全相关，是因为有人可能会蓄意使数据或服务失效，以此来拒绝对数据或服务的访问。系统设计经常采用一个统计模型来分析期望的系统使用模式，当该统计模型起作用时，有多种机制能确保系统的可用性。有人也许可以操控系统的使用（或者操控用以控制系统使用的参数，如网络通信量），使得现有统计模型的假设不再有效。这意味着用于保证资源和数据可用性的机制没有工作在它们的设定环境中，其结果通常是这些机制将失效。

例 1.3　假设 Anne 攻破了某家银行用于提供银行账目结算服务的辅服务器，那么当任何人向该服务器查询信息时，Anne 可以响应任意信息。通过和银行的结算主服务器联系，可使商户的支票生效。如果商户得不到响应，就会要求辅服务器提供数据。Anne 的同伙阻断了商户与结算主服务器之间的通信，所以商户的所有服务请求都会转到辅服

务器。无论 Anne 的实际账目余额为多少，她都绝不会拒绝处理商户的这些支票。注意：若银行只有一台服务器（主服务器），那么这样的方案是行不通的，商户将无法使其支票生效。

企图破坏系统的可用性称为拒绝服务攻击，这可能是最难检测的攻击，因为这要求分析者能够判断异常的访问模式是否可以归结于对资源或环境的蓄意操控。统计模型的本质注定使得这种判断非常复杂。即使模型精确地描述了环境，异常事件也仅仅对统计学特性起作用。蓄意使资源失效的企图可能仅仅看起来像异常事件，或者可能就是异常的事件。但在有些环境下，这种企图甚至可能表现为正常事件。

1.5　信息系统安全策略

在考虑如何确保信息系统的安全性之前，首先必须清楚什么样的信息系统可以称得上是安全的。从某种意义上说，符合安全策略要求的信息系统就是安全的信息系统。那么，如何定义安全策略呢？这是本节需要解答的问题。安全策略与安全威胁密切相关，所以，本节的介绍就从安全威胁开始。

1.5.1　信息安全威胁

威胁是对安全的潜在破坏，这种破坏不一定要实际发生。破坏可能发生的这个事实意味着必须防止（或预防）那些可能导致破坏发生的行为，这些行为称为攻击，行为的完成者或导致行为完成的人称为攻击者。

有三种安全服务——机密性、完整性和可用性——能减少系统安全的威胁。Shirey 将威胁分为四大类：泄露，即对信息的非授权访问；欺骗，即接受虚假数据；破坏，即中断或妨碍正常操作；篡夺，即对系统某些部分的非授权控制。这四大类威胁涵盖了很多种常见的威胁。因为威胁是普遍存在的，所以，这些问题将反复出现在信息系统安全的整个学习过程中。以下将对每种威胁进行简单介绍。

嗅探，即对信息的非法拦截，是某种形式的信息泄露。嗅探一般是被动的，即某些实体仅仅是窃听（即读取）消息，或者仅仅浏览文件或系统信息。搭线窃听即被动搭线窃听是一种监视网络的嗅探形式（称它为搭线窃听，是因为线路构成了网络，在不涉及物理线路时也使用这个术语）。机密性服务可以对抗这种威胁。

篡夺或更改是对信息的非授权改变。篡改的目的可能是欺骗，在欺骗过程中，一些实体要根据修改后的数据来决定采取什么样的动作，或者不正确的信息会被当做正确的信息而接收和发布。如果被修改的数据控制了系统操作，就会产生破坏和篡夺的威胁。与嗅探不同的是，篡改是主动性的，其起因是某实体对信息做出变更。主动搭线窃听是篡改的一种形式，在窃听过程中，传输于网络中的数据会被更改，“主动”使得主动搭

线窃听有别于嗅探（被动搭线窃听）。中间人攻击就是一种主动搭线攻击的例子：入侵者从发送者那里读取消息，再将（可能修改过的）不同版本的消息发往接收者，并希望接收者和发送者不会发现中间人的存在。完整性服务能对抗这种威胁。

伪装或电子欺骗，即一个实体被另一个实体假冒，是兼有欺骗和篡夺的一种手段。这种攻击引诱受害者相信与之通信的是另一个实体。例如，如果某用户试图通过 Internet 登录一台计算机，但实际却进入了另一台自称是前者的机器，这样，用户就被欺骗了。类似的情况还包括：如果某用户尝试去读取一个文件，但是攻击者却给他准备了一份不同的文件，这也是一种欺骗。伪装可以是被动攻击（用户并不想认证接收者的身份，而直接访问它），但通常是主动攻击（冒充者发出响应来误导用户，以隐瞒身份）。尽管伪装的主要目的是欺骗，但攻击者也常常通过伪装来冒充被授权的管理员或控制者，以篡夺系统的控制权。完整性服务（在此称为“认证服务”）能对抗这种威胁。

某些形式的伪装是允许的，如委托行为，即一个实体授权另一个实体代表自己来行使职责。伪装与委托之间存在重大的差别。如果 Susan 委托 Thomas，让他代表自己，即 Susan 赋予 Thomas 执行特别操作的许可，就像是他自己在执行这些操作一样。各方都知道这种委托关系。Thomas 不会假装成 Susan；相反，他会说：“我是 Thomas，并且我有权代表 Susan 来做这件事。”如果有人询问 Susan，他也会证实 Thomas 说的是真的。然而，如果是伪装攻击，则 Thomas 会假冒成 Susan，没有任何一方（包括 Susan）会发现这种伪装，Thomas 会说：“我是 Susan。”如果任何人发现与之交易的是 Thomas，并就此事询问 Susan 的话，Susan 会否认他曾授权 Thomas 来代表自己。就安全而言，伪装是一种破坏安全的行为，而委托不是。

信源否认，即某实体欺骗性地否认曾发送（或创建）过某些信息，是某种形式的欺骗。例如，假设一个顾客给某供货商写信，说同意支付一大笔钱来购买某件产品。供货商将此产品送过去并索取费用。这个顾客却否认曾经订购过该产品，并且根据法律，他有权不支付任何费用而可扣留这件主动送上门的产品。即顾客否认了信件的来源。如果供货商不能证明信件来自于这位顾客，那么攻击就成功了。这种攻击的一种变形是：用户否认曾创建过特定的信息或实体，如文件等。完整性机制能应付这种威胁。

信宿否认，即某实体欺骗性地否认曾接收过某些信息或消息，这也是一种欺骗。假设一个顾客预定了一件昂贵的商品，但供货商要求在送货前付费。该顾客付了钱，然后供货商送来商品，接着顾客再一次询问供货商何时可以送货。如果顾客以前已经接收过货物，那么这次询问就构成一次信宿否认攻击。为了免于攻击，供货商必须证明顾客确实接收过货物，尽管顾客在否认。完整性机制和可用性机制都能防止这种攻击。

拖延，即暂时性地阻止某种服务，这是一种篡夺攻击，尽管它能对欺骗起到支持的作用。通常消息的发送服务需要一定的时间，如果攻击者能够迫使消息发送所花的时间多于前面所需的时间，那么攻击者就成功地拖延了消息发送。这要求对系统控制结构的

操控，如对网络部件或服务器部件的操控，因此它是一种篡夺。如果某实体等待的认证信息被拖延了，它可能会请求辅服务器提供认证。即使攻击者可能无法伪装成主服务器，但是他可能伪装为辅服务器以提供错误的信息。可用性机制能缓解这种威胁。

拒绝服务，即长时间地阻止服务，这也是一种篡夺攻击，尽管它常与其他机制一起被用于欺骗。攻击者采用的方法是阻止服务器提供某种服务。拒绝可能发生在服务的源端（即通过阻止服务器取得完成任务所需的资源），也可能发生在服务的目的端（即阻断来自服务器的信息），还可能发生在中间路径（即通过丢弃从客户端或服务器端传来的信息，或者同时丢弃这两端传来的信息）。拒绝服务产生的威胁类似于无限的拖延。可用性机制能对抗这种威胁。

拒绝服务或拖延可能是直接攻击引起的，也可能是由与安全无关的问题导致的。从我们的观点来看，原因与结果是重要的，而隐藏于原因与结果下的目的却并不重要。如果拖延或拒绝服务破坏了系统安全，或者是导致系统遭到破坏的一系列事件中的一部分，那么可将它视为破坏系统安全的一种企图。这种企图可能并非蓄意的破坏，事实上，它可能是环境特征的产物，而非攻击者的特定行为。

1.5.2　策略与机制

策略与机制之间的差别对于安全的研究非常关键。

定义 1.1　安全策略是对允许什么、禁止什么的规定。

定义 1.2　安全机制是实施安全策略的方法、工具或规程。

机制可以是非技术性的，如在修改口令前总要求进行身份认证。实际上，策略经常需要一些技术无法实施的过程性机制。

举个例子，假如某所大学的计算机科学实验室有一项策略：禁止任何学生复制其他学生的作业文件。计算机系统提供防止其他人阅读用户文件的机制。Anna 没有使用这些机制来保护她的作业文件，并且被 Bill 复制了作业。这就发生了安全破坏，因为 Bill 违反了安全策略。Anna 未能保护好她的文件，但她并未授权 Bill 复制她的作业。

在这个例子中，Anna 本可以容易地保护好她的文件。在其他环境中，这样的保护就不容易了。例如，Internet 只提供最基本的安全机制，这不足以保护传送于网络中的信息。然而，像记录口令和其他敏感信息这样的行为违反了大多数站点隐含的一条安全策略（特别地，口令是用户的保密特性，任何人都不能记录）。

可以用数学方法表达策略，将其表示为允许（安全）或不允许（不安全）的状态列表。为达到这个目的，可假设任何给定的策略都对安全状态和非安全状态做了公理化描述。实践中，策略极少会如此精确，通常策略使用文本语言描述什么是用户或工作人员允许做的事情。这种描述的内在歧义性导致某些状态既不能归于“允许”类，也不能归

于“不允许”类。例如，考虑上面讨论过的作业策略。如果某人查看另一用户的目录，但不复制作业文件，这破坏安全了吗？问题的答案取决于特定单位的习惯、规则、规章和法律，这些都不是我们关注的焦点，而且它们也会随时间而改变。

当两个不同的单位进行通信或合作时，它们组成的实体会使用一种新的安全策略，并且这种策略要建立在这两个单位的安全策略之上。如果这两个单位的策略不一致，则必须由这两个单位之一或这两个单位共同决定复合单位的安全策略。这种不一致性经常表现为对安全的破坏。例如，如果把专利文件发给学校，公司的保密策略将会与大多数学校的更开放的策略相冲突。学校和公司必须开发共同的安全策略以满足双方的需求，这才能产生相容策略。当两个单位通过独立的第三方（如 Internet 服务提供商）进行通信时，这种情形下的复杂度就会陡然增加。

1.5.3　安全的目的

如果给定描述“安全”和“非安全”行为的安全策略规范，安全机制就能够阻止攻击、检测攻击或在遭到攻击后恢复工作。可以组合使用安全机制，也可以单独使用安全机制。

阻止意味着攻击的失败。例如，如果有人企图通过 Internet 闯入某台主机，但那台主机并没有连入 Internet，这就阻止了攻击。一般地，阻止涉及对机制的实现，要求所实现的机制是使用者无法逾越的，同时也必然是通过正确的、不能滥改的方法实现的，使得攻击者不能用改变机制的方法来攻破机制。阻止机制往往非常笨重，它会干扰系统的使用，甚至达到阻碍系统正常使用的程度。但一些简单的阻止机制，如口令等机制已被广泛接受（设置口令的目的是要防止非授权用户访问系统）。阻止机制能防止系统的不同部分遭受攻击。一旦阻止机制被合理实现，该机制所保护的资源就无须因为安全问题而受到监控，至少从理论上说是这样的。

当不能阻止攻击时，检测是最有用的，同时它也体现出防范措施的有效性。检测机制通常基于这样的理念：攻击总会发生，检测的目的就是要判定攻击是正在进行中还是已经发生。检测机制会基于结果给出报告。尽管有可能无法阻止攻击，但可监视攻击，从而能采集有关攻击的性质、严重性和结果的数据。典型的检测机制要监视系统的各个方面，寻找预示攻击的行为或信息。这种机制的一个很好的例子是：当用户输入三次错误的口令后就发出警告。登录程序还可继续，但是系统日志中的一条报错信息将报告这次不寻常的多次输入错误口令的事件。检测机制不能阻止对系统的攻击，这是一种严重的缺陷。受检测机制保护的资源会因安全问题而被持续不断地或周期性地监控。

恢复有两种形式。第一种是阻断攻击，并且评估、修复由攻击造成的任何损害。例如，若攻击者删除了一份文件，那么某恢复机制应能从备份介质中恢复该文件。实践中，恢复要远远复杂得多，因为每种攻击都有其独特的性质。因此，很难完整地刻画出

任何一种损害的类型和程度。此外，攻击可能会再次发生，所以恢复的功能应包括辨识和消除攻击者赖以闯入系统的系统脆弱性。在某些情况下，报复（通过攻击攻击者的系统或通过合法渠道让攻击者对其行为负责）也是恢复的一部分。在所有这些情况下，系统的机能都因攻击而受到压制。依据定义，恢复还应具备还原正确操作的功能。

第二种恢复方式要求在攻击发生后系统仍能继续工作。由于计算机系统的复杂性，这类恢复难以实现。这类恢复机制同时利用容错技术和安全技术，一般用于可靠性非常关键的系统。与第一种方法不同，在任何时候这种系统都不会在功能上出错，而只会将系统不重要的功能禁用，当然，这类恢复常常会以一种较弱的方式出现。借助这种方法，系统可自动检测错误的功能，并改正（或试图改正）该错误。

1.5.4　安全策略的意义

安全策略为系统或系统集定义“安全”。安全策略在本质上可以是非形式化的，也可以是高度数学化的。

如果将计算机系统看成一种有限状态自动机，该自动机有一套完成状态改变的转换函数，那么我们有以下的定义。

定义 1.3　安全策略是一种声明，它将系统的状态分成两个集合：已授权的（即安全的）状态集合和未授权的（即不安全的）状态集合。

安全策略设置了可以定义安全系统的环境。在某种策略下安全的系统不一定在另一种策略下也安全。更确切的意义可由以下定义表示。

定义 1.4　安全系统是一种始于已授权状态但不能进入未授权状态的系统。

考虑如图 1.3 所示的有限状态自动机，它由 4 个状态和 5 个转换组成。安全策略将这些状态分成两个集合，一个是已授权的状态集合 $A=\{s_1, s_2\}$，另一个是未授权的状态集合 $UA=\{s_3, s_4\}$。这个系统是不安全的，原因在于无论它开始于哪个授权状态，它都可以进入一个未授权的状态。当然，如果删除从 s_1 到 s_3 的边，那么系统就是安全的了，因为从已授权的状态不能进入未授权的状态。

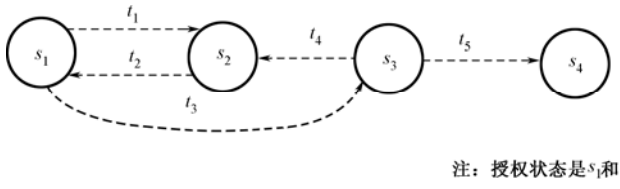


图 1.3　一个简单的有限状态自动机

定义 1.5　如果系统进入一个未授权状态，则称发生了一次安全破坏。

前面非形式化地讨论了与安全相关的三个基本属性（机密性、完整性和可用性）。下面给出确切的定义。

定义 1.6　设 *X* 是实体的集合，并设 *I* 是某种信息。如果集合 *X* 中的成员不能获取信息 *I*，那么称 *I* 关于 *X* 具有机密性。

机密性意味着信息不能透露给某些实体，但它可以透露给另外的一些实体。集合 *X* 的成员关系是隐式定义的。例如，当我们说一个文档是保密的时，是指某些实体可以访问该文档，而没有授权访问该文档的实体就组成了集合 *X*。

定义 1.7　设 *X* 是实体的集合，并设 *I* 是某些信息或某种资源。如果集合 *X* 中的所有成员都信任 *I*，那么称 *I* 关于 *X* 具有完整性。

这个定义并非表面上的这么简单。*X* 的成员不仅信任信息本身，而且相信信息 *I* 的传输和保存没有引起信息本身及其可信性的改变（这种情况通常称为数据完整性）。如果 *I* 是关于某物的来源或某身份标识的信息，那么 *X* 的成员信任信息是正确的且未经改变的（这种情况通常称为来源完整性或可认证性）。另外，*I* 也可能是资源而不是信息，这时，完整性就指资源工作正常（符合规范）。如同机密性的定义，*X* 的成员关系也通常是隐式定义的。

定义 1.8　设 *X* 是实体的集合，并设 *I* 是某种资源。如果集合 *X* 中的所有成员都可以访问 *I*，那么称 *I* 关于 *X* 具有可用性。

根据集合 *X* 中的成员需求、资源性质或资源使用地点的不同，定义 1.8 中“访问”的确切定义也会有所不同。如果一台售书服务器要花 1 小时来完成一项购书请求的服务，这也许可以满足客户的“可用性”的需求。但是，如果一台医疗信息服务器要花费超过 1 小时的时间来响应请求，以便确认某种麻醉剂会产生哪些过敏反应，那么这将不能满足急诊室“可用性”的需求。

安全策略涉及机密性、完整性和可用性等问题的所有相关方面。关于机密性，安全策略要确定这样的状态：在这些状态下信息泄露到未授权接收该信息的实体中。这不仅包括权限的泄露，而且包括没有权限泄露的非法信息传输，后者称为信息流。而且，安全策略必须处理授权的动态变化，因此它应包含一个时态元素。例如，公司的合同工可以在保密协议有效期间访问某些专利信息，一旦保密协议过期，该合同工将不能再访问这些信息。这种类型的安全策略通常称为保密策略。

关于完整性，安全策略既要指定可以改变信息的已授权方法，也要指定能够改变信息的已授权实体。授权可能由多种关系衍生，而且可能受到外部影响的约束。例如，在许多事务中，“职责分离”原则禁止一个实体单独完成一项事务。安全策略中描述可改变数据的条件和方式的部分称为完整性策略。

关于可用性，安全策略描述了必须提供的服务。它可以给出服务的参数，指定服务可被访问的范围。例如，规定浏览器可以下载 Web 页面，但不能下载 Java 小程序。这种策略也可能需要用到服务等级，如服务器必须在请求提出 1 分钟之内提供认证数据。可用性与服务质量（QoS）直接相关。

安全策略的声明可以形式化地描述系统所要求的属性。如果要求某个系统为可证明是安全的，那么设计者和实施者必须能够使用形式化声明来证明系统满足所要求的属性。如果形式化声明是不必要的或不可行的，则分析者可以测试所要求属性对某些输入的集合是否成立。

在实践中，有一类不太形式化的安全策略，它首先定义了已授权状态的集合。通常，这种安全策略假设读者明白策略颁布的环境——尤其是法律、组织性策略和其他环境因素。然后此安全策略描述用于定义“已授权用户”和“已授权使用”的行为、操作和授权。

例 1.4　某大学不允许作弊，作弊的定义包括抄袭他人的作业（经过许可或未经许可）。有一门计算机科学课程要求学生在系里的计算机上完成作业。有个学生发现另一个学生没有对她的作业文件进行读保护，并抄袭了该作业。那么哪个学生破坏了安全？

第二个学生并没有破坏安全，尽管她没有保护好她的作业。此安全策略没有要求使用任何操作来防止文件被读。虽然她过于相信别人，但安全策略没有对此禁止。所以，第二个学生没有破坏安全。

第一个学生破坏了安全。此安全策略不允许抄袭作业，但这个学生恰恰这么做了。无论安全策略是明确地规定“包含作业的文件是不能被复制的”，还是仅仅规定“用户必须受校规的约束”，这都不会影响该结论的判定。对于后者，规则中有一条是禁止作弊。如果安全策略没有说明这种情况，最合理的解释就是因为计算机科学系是学校的一部分，所以该策略也应遵守校规。

如果有人反驳说第一个用户有复制文件的权力，因此他的行为是允许的，那么这就混淆了机制和策略的区别，这两者的区别是明显的。

定义 1.9　安全机制是实施安全策略的某些部分的组件或规程。

例 1.5　在前面的例子中，策略声明任何学生不得抄袭他人的作业。实施这种策略的一种安全机制就是文件访问控制：如果第二个学生设置权限来防止第一个学生读她的作业文件，则第一个学生就不能复制该文件。

例 1.6　一家公司的安全策略规定：与某种特定产品相关的信息是专利信息，且不能脱离公司的控制。公司将备份磁带保存在城市银行的保险库里（这样做通常是为了防止计算机设备遭到完全破坏）。公司必须确保只有被授权的员工才能访问备份磁带，即

便磁带是离线存储的。这样，银行对保险库的访问控制、对用于从银行中输入/输出磁带的过程的控制就是安全机制。需要注意的是：这些机制不是建立在计算机之上的技术控制。规程控制或运行控制也同样是安全机制。

安全策略有时是隐含定义而不是明确定义的。这往往会带来混乱，尤其是当策略由多种机制定义时。这样的定义可能模棱两可。例如，如果某些机制禁止特定行为，而其他机制却允许这些行为。这样的策略就会带来混乱，须避免。

例 1.7　UNIX 操作系统最初是为规模较小的研究团体开发的，它采用了充分的机制来防止用户意外地破坏他人的文件。例如，用户 ken 不能删除用户 dmr 的文件（除非 dmr 恰当地设置了该文件和包含该文件的文件夹）。在这样一个友好的环境下，其隐含的安全策略是“不要删除或破坏他人的文件，未加保护的任何文件都是可读文件”。

一旦 UNIX 操作系统用于科研、商业机构或政府部门，那么上述安全策略就不充分了。例如，有些文件对于个别用户（而不是用户组）是不可访问的。毫无疑问，这时上述安全机制也是不充分的。

出于分析的目的，区分策略与策略的抽象描述是很重要的。

定义 1.10　安全模型是表达特定安全策略或安全策略集合的模型。

模型抽象出与分析相关的多种细节。很少有分析讨论专门的策略，因为许多策略体现出多种细节特征，所以通常分析都侧重于策略的这些特征。而且具有这些特征的策略越多，策略的分析就越有效。根据 HRU 结果，没有单一的能够涵盖所有策略的非平凡分析，但是约束安全策略的分类可以有效地对多种类型的策略进行分析。

1.5.5　安全策略的类型

不同的场合有不同的机密性、完整性和可用性的需求级别，不同场合下的策略描述了该特定场合的需求。

定义 1.11　军事安全策略（又称政府安全策略）是以提供机密性为主要目的的安全策略。

该名称源于军事上保密信息的需求，如一艘军舰的出航日期。虽然完整性和可用性也是重要的，使用这类策略能够减少这两者的损失（如不使用通过计算机网络发送的命令），但是如果机密性受到危害，那么后果将是灾难性的，因为敌方可能会制订对抗计划（而己方却没有察觉机密性已受到危害）。

机密性是隐私的一部分，而隐私权是得到许多政府实体的法律认可的（如美国的隐私法和瑞典的相关法律）。除了限制政府实体可以合法地获取何种个人信息外，这些法

律也限制了这些信息的泄露与滥用。未经授权的隐私泄露可能会导致监禁或罚款，而且这样的泄露也会破坏公众对政府的信任和尊重，进而使公众不再向这些泄密机构提供类似的信息。

定义 1.12　商业安全策略是以提供完整性为主要目的的安全策略。

该名称源于商业机构防止数据被篡改的需求，因为它们承受不起这样的危害。例如，如果银行计算机的机密性受到了危害，可能会导致客户账户资金的泄露。这必然会使银行陷入尴尬的境地，客户也可能因此而将其业务转移到其他银行。如果存有账户信息的计算机的完整性受到危害，那么客户账户资金可能会被改变，从而导致财务上的毁灭性后果。

一些完整性策略使用“事务”这个概念，正如数据库规范一样，它们要求使用可以保持数据库状态的一致性的操作。这些策略称为面向事务的完整性策略，它们对于需要数据库一致性的机构而言是非常重要的。

例 1.8　为了实现客户将钱从一个账户转移到另一账户的功能，银行使用了一种良定义的事务，该事务包含两个不同的部分：钱先从前面的账户借出，再存入后一个账户。如果两个事务不是一同完成的，客户就会损失金钱。如果使用良定义的事务处理，即便事务中断（不管是发生在事务开始前还是事务完成前），数据库的状态仍然是一致的。因此，银行安全策略的一部分就是所有的事务必须是良定义的。

以上策略的信任角色凸显出这些策略间的差异。机密性策略没有对客体设置信任。就所关注的策略而言，客体可以是内容正确的报告，也可以是从伊索寓言里摘录的故事。策略描述规定了客体是否可以被泄露，但它并没有指出是否应该信任客体。

与机密性策略相反，完整性策略表明可在多大程度上信任客体。假设某种信任级别是正确的，完整性策略规定了主体可以怎样处理客体。但棘手的问题是如何指派信任级别。例如，如果站点获取了某程序的一个新版本，那么这个程序的完整性应该是高（表示站点信任程序的新版本）还是低（表示该站点还不信任这个新程序）？还是应该介于两者之间（由于程序是供应商提供的，并没有像老版本程序一样在本地站点经历了完全的测试）呢？这就使得完整性策略比机密性策略模糊得多。机密性级别的指派基于划分者希望其他人知道什么，而完整性级别的指派基于划分者对信息可信度的主观评判。

有两个描述策略的术语与安全需求相关，因为它们会在其他地方出现，现在就对其进行定义。

定义 1.13　机密性策略是仅处理机密性的安全策略。

定义 1.14　完整性策略是仅处理完整性的安全策略。

虽然机密性策略和军事策略都是处理机密性的，但机密性策略并不处理完整性，而军事策略却要处理完整性。完整性策略和商业策略的区别与此相似。

1.6　信息系统访问控制

获取对信息系统的访问权是攻击者实施安全攻击的重要目的之一，相应地，安全防御的重要目的之一就是防止攻击者对信息系统进行非法访问。访问控制是确保信息系统安全的重要措施之一。

1.6.1　访问控制矩阵

描述一个保护系统的最简单框架是使用访问控制矩阵模型，这个模型将所有用户对文件的访问权限存储在矩阵中。访问控制矩阵模型最早由 B.Lampson 于 1971 年提出，Graham 和 Denning 对它进行了改进，这里将使用他们的模型。

客体集合 O 是指所有被保护实体的集合（所有与系统保护状态相关的实体）。主体集合 S 是所有活动对象的集合，如进程和用户。所有的权限的类型用集合 R 来表示。在访问控制矩阵模型中，客体集合 O 和主体集合 S 之间的关系用带有权限的矩阵 $\boldsymbol{A}$ 来描述，$\boldsymbol{A}$ 中的任意元素 $\alpha[s, o]$ 满足 $s \in S$，$o \in O$，$\alpha[s, o] \subseteq R$。元素 $\alpha[s, o]$ 代表的意义是主体 s 对客体 o 具有访问权限 $\alpha[s, o]$。

所有保护状态的集合可以用一个三元组（$S, O, \boldsymbol{A}$）来表示。图 1.4 给出了一个系统的保护状态的例子。在这个例子中，进程 1 可以对文件 1 进行读、写操作，对文件 2 进行读操作；进程 2 可以对文件 1 进行添加并且可以读文件 2；进程 1 可以通过向进程 2 写数据的方法（如管道）和进程 2 通信，当然，进程 2 可以读取进程 1 传给它的数据。每个进程都是本进程的拥有者，同时进程 1 是文件 1 的拥有者，进程 2 是文件 2 的拥有者。需要注意的是进程既是客体又是主体，这使得进程既可以作为操作者又可以作为被操作的对象。

	文件1	文件2	进程1	进程2
进程1	read write own	read	read write execute own	write
进程2	append	read own	read	read write execute own

注：权限集为{read, write, execute, append, own}，系统有两个进程和两个文件

图 1.4　一个系统的保护状态的例子

不同的系统有不同的对权限意义的解释。一般来说，从文件读取、写到文件和添加数据到文件这些操作的意义都是很明显的。但是，“从进程读数据”这种操作代表什么意义呢？这和系统的实现有关，它可以代表从该进程获取一个消息或只是简单地查看进

程当前的状态（和调试器一样）。系统操作所涉及的客体不同也导致权限的意义有所不同。理解访问控制矩阵模型的关键点在于它只是描述保护状态的抽象模型，如果要谈到某个具体的访问控制矩阵的意义，则必须和系统的具体实现联系起来。

对某个客体的拥有权是一种特殊的权限。在大多数系统中，某个客体的创造者拥有对该客体的某些优先权：增加或删除其他用户对该客体的权限。在如图 1.4 所示的例子中，进程 1 可以修改 ***A***[*x*，文件 1]的值，这里的 *x* 是任意的主体。

例 1.9　UNIX 操作系统对于文件访问定义了读、写和执行权限，这些权限的意义是非常清楚的。然而对于目录，这些权限的意义就有所不同了。一个进程访问目录时，读的权限意味着进程可以列出目录中的所有文件；写的权限意味着进程可以在该目录下创建文件；执行的权限意味着进程可以访问该目录下的文件和子目录。对于进程，这些权限的意义也有所不同。当一个进程 F 与另一个进程 G 交互时，读的权限意味着进程 F 可以从进程 G 获取消息；写的权限意味着进程 F 可以向进程 G 发送消息；执行的权限意味着进程 F 可以将进程 G 当做子进程运行。

系统的超级用户对任何（本地）文件都有访问权限，不论该文件的所有者是否给予了超级用户这些权限。从实际效果来看，这样的超级用户相当于系统中所有文件的拥有者，但是超级用户拥有权限的解释也必须遵循上面所述的限制。例如，超级用户也不能使用操作文件的系统调用和系统命令来操作目录；超级用户也必须使用适当的系统调用和系统命令来创建、删除和重命名文件。

访问控制矩阵中的客体一般意味着文件、设备或者进程，但客体其实可以是小到进程之间发送的一条消息，也可以大到整个系统。图 1.5 显示了一个局域网（LAN）中三个不同系统的访问控制矩阵。在这个例子中，权限对应着不同的网络协议：“own”权限对应着在服务器上创建一个服务的权限；“ftp”意味着使用文件传输协议（FTP）来访问文件；“nfs”意味着使用网络文件系统协议（NFS）来访问文件；“mail”权限意味着可以使用简单邮件传输协议（SMTP）来发送邮件。

	telegraph	nob	toadflax
telegraph	own	ftp	ftp
nob		ftp nfs mail own	ftp nfs mail
toadflax		ftp mail	ftp nfs mail own

注：权限集为{ftp，mail，nfs，own}

图 1.5　一个 LAN 中三个不同系统的访问控制矩阵

在如图 1.5 所示的例子中，客体 telegraph 是个人计算机，它上面运行了一个 FTP 的客户端，可以通过这个 FTP 客户端访问其他两个系统。不过 telegraph 没有提供任何服务，其他两个系统都无法访问它。客体 nob 向不包括客体 toadflax 在内的一些客户提供 NFS 服务，nob 和 toadflax 之间可以互发邮件，它们也都提供了 FTP 服务，以便其他系统能访问它们。

在更微观的层次，访问控制矩阵也可以为计算机程序语言建模。在这种情况下，客体指程序中的变量，主体是程序中的进程或模块。考虑一个需要事件同步的程序，程序中的一个模块提供了对这个模块内私有计数器进行增加（inc_ctr）和减少（dec_ctr）的函数。子程序 manager 可以调用这些函数。这样，访问控制矩阵如图 1.6 所示，“+”和“–”的权限分别对应增加计数器和减少计数器。“call”权限意味着可以调用某个函数（子程序），子程序 manager 可以调用自己。

计算器	inc_ctr	dec_ctr	manager	
inc_ctr	+			
dec_ctr	-			
manager		call	call	call

注：权限集为{+, -, call}

图 1.6　访问控制矩阵

在上面的例子中，访问控制矩阵中的元素是权限。当然，这些元素也可以代表确定某个特定状态下的权限的函数，函数的参数可以是一些历史数据（如以往的访问数据）、当天的时间、其他主体对该客体拥有的权限等。常见的这类函数是用于限定 Bernstein 条件的锁定函数。锁定函数可以保证一个进程在写文件时，其他进程不能访问该文件，当写的过程结束之后，其他进程才可拥有访问该文件的权限。

1.6.2　访问控制的类型

安全策略可以单独地或混合地使用两种类型的访问控制：其一，访问控制取决于拥有者的判断力；其二，操作系统控制访问，且拥有者不能超越这些控制。

第一种访问控制的类型基于用户身份，是应用最为广泛的一种类型。

定义 1.15　如果作为客体的拥有者的个人用户可以设置访问控制属性来许可或拒绝对客体的访问，那么这样的访问控制称为自主访问控制（DAC，Discretionary Access Control）。

自主访问控制的访问权限基于主体和客体的身份。身份是关键，客体的拥有者通过允许特定的主体进行访问来限制对客体的访问。拥有者根据主体的身份来规定限制，或者根据主体的拥有者来规定限制。

例 1.10　假设一个小孩有本日记。她控制着对日记的访问权，因为她可以决定谁能阅读（授予读访问权限）或谁不能阅读（拒绝读访问）。她允许妈妈读，但其他人不行。这是自主访问控制，因为对日记的访问由客体（日记）的拥有者（小孩）确定。

访问控制的第二种类型是基于系统的授权，与个人用户无关。

定义 1.16　如果只有系统才能控制对客体的访问，而个人用户不能改变这种控制，那么这样的访问控制称为强制访问控制（MAC，Mandatory Access Control）。

操作系统实施强制访问控制。主体和客体的拥有者都不能决定访问的授权。通常，系统机制通过检查主体与客体相关的信息来决定主体是否可以访问客体。规则描述允许访问的条件。

例 1.11　法律允许法庭在未经拥有者授权的情况下查看其驾驶记录。这是强制访问控制，因为记录的拥有者不能控制法庭对信息的访问。

定义 1.17　创建者控制的访问控制（ORCON 或 ORGCON）基于客体（或客体所包含信息）的创建者进行数据访问。

这种控制的目的是允许文件（或它所包含的信息）的创建者控制信息的分发，而文件的拥有者则没有控制谁能访问该文件的控制权。

例 1.12　一家以开发嵌入式系统而著称的甲公司与乙公司签订了一份合同，乙公司以其微代码设计能力而著称。合同要求乙公司为某种处理器开发微代码设计语言，该处理器用于高性能嵌入式系统。甲公司给了乙公司一份处理器规范的副本。合同的条款要求乙公司在透露处理器信息给子承包商前，要先获得授权。这就是一种创建者控制的访问机制，因为尽管乙公司拥有包含规范的文件，但它不能让他人接触此信息，除非得到创建者甲公司的允许。

1.6.3　贝尔–拉普杜拉访问控制模型

1973 年，贝尔（D.E. Bell）和拉普杜拉（L.J. LaPadula）提出了一个可证明的安全系统的数学模型，这就是贝尔–拉普杜拉模型（Bell&LaPadula 模型），简称贝–拉模型（BLP 模型）。在随后的几年，该模型得到了进一步的充实和完善。贝尔和拉普杜拉在 1976 年完成的研究报告给出了贝–拉模型的最完整表述，其中包含模型的形式化描述和非形式化说明，以及模型在 Multics 系统中实现的解释。

贝–拉模型是根据军事安全策略设计的，它要解决的本质问题是对具有密级划分的信息的访问进行控制。

贝−拉模型是一个状态机模型，它定义的系统包含一个初始状态 z_0 和由一些三元组（请求，判定，状态）组成的序列，三元组序列中相邻状态之间满足某种关系 W。如果一个系统的初始状态是安全的，并且三元组序列中的所有状态都是安全的，那么这样的系统就是一个安全系统。

贝−拉模型定义的状态是一个四元组（b, $\boldsymbol{M}$, f, H），其中，b 是当前访问的集合，当前访问由三元组（主体，客体，访问方式）表示，是当前状态下允许的访问；$\boldsymbol{M}$ 是访问控制矩阵；f 是安全级别函数，用于确定任意主体和客体的安全级别；H 是客体间的层次关系。抽象出的访问方式有 4 种，分别是只可读 $\underline{r}$、只可写 $\underline{a}$、可读写 $\underline{w}$ 和不可读写（可执行）$\underline{e}$。主体的安全级别包括最大安全级别和当前安全级别，最大安全级别通常简称为安全级别。以下特性和定理构成了贝−拉模型的核心内容。

简单安全特性（ss−特性）：如果（主体，客体，可读）是当前访问，那么一定有：

$$\text{level(主体)} \geqslant \text{level(客体)}$$

其中，level 表示安全级别，$\geqslant$ 表示支配关系。

星号安全特性（*−特性）：在任意状态，如果（主体，客体，方式）是当前访问，那么一定有：

（1）若方式是 $\underline{a}$，则 level(客体) $\geqslant$ current-level(主体)；

（2）若方式是 $\underline{w}$，则 level(客体) = current-level(主体)；

（3）若方式是 $\underline{r}$，则 current-level(主体) $\geqslant$ level(客体)。

其中，current-level 表示当前安全级别。

自主安全特性（ds−特性）：如果（主体-i，客体-j，方式-$\underline{x}$）是当前访问，那么，方式-$\underline{x}$ 一定在访问控制矩阵 $\boldsymbol{M}$ 的元素 M_{ij} 中。

与 ds-特性处理自主访问控制相对应，ss-特性和*-特性处理的是强制访问控制。自主访问控制的权限由客体的属主（即拥有者）自主确定，强制访问控制的权限由特定的安全管理员代表系统确定，由系统强制实施。

基本安全定理：如果系统状态的每一次变化都能满足 ss-特性、*-特性和 ds-特性的要求，那么，在系统的整个状态变化过程中，系统的安全性是不会被破坏的。

贝−拉模型中的安全级别包含等级分类和非等级类别两种元素，等级分类可以与整数相对应，非等级类别可以与集合相对应。安全级别可由以下定义表示。

定义 1.18　在贝−拉模型中，实体 E 的安全级别 level(E)是一个二元组(L, C)，其中，L 是等级分类，可与一个整数相对应，C 是非等级类别，可以与一个集合相对应。

例 1.13　贝–拉模型的安全级别的等级分类元素可以表示现实应用中的以下密级：

非密、秘密、机密、绝密

它们可以分别用以下整数表示：

1、2、3、4

安全级别的非等级类别可以表示现实应用中的以下集合：

{财务处}、{科研处}、{教务处}、{财务处，科研处}、{财务处，教务处}、……

贝–拉模型中的安全级别的支配关系由以下定义描述。

定义 1.14　设在贝–拉模型中，实体 E_1 和 E_2 的安全级别分别为

$$\text{level}(E_1) = (L_1, C_1),\ \text{level}(E_2) = (L_2, C_2)$$

若用 $\geqslant$ 表示支配关系，那么：

$$\text{level}(E_1) \geqslant \text{level}(E_2)$$

当且仅当：

$$L_1 \geqslant L_2 \text{ 且 } C_1 \supseteq C_2$$

1.6.4　系统保护状态

保护系统描述了在哪些条件下被保护的系统是安全的。访问控制矩阵是描述保护系统的经典方法。访问控制矩阵最早源于对操作系统和数据库的研究，它使用一个矩阵来描述访问权限。

系统状态是由所有内存、二级缓存、寄存器和系统中其他设备的状态构成的集合，这个集合中涉及安全保护的子集称为保护状态，访问控制矩阵是用于描述当前保护状态的工具。

假设 P 是所有可能的保护状态的集合，Q 是 P 的一个子集，Q 代表系统的合法状态。这样，如果系统在任何时候的状态都在集合 Q 中，则系统是安全的；如果系统的当前状态是处于 $P–Q$ 中（$P–Q$ 表示由所有在集合 P 中，但不在集合 Q 中的元素构成的集合），那么系统是不安全的。描述系统状态的有用之处是要列出处于集合 Q 中的系统状态的特征，实现安全是要保证系统的状态总是属于集合 Q 的。描述集合 Q 中的系统状态的特征是安全策略的工作，阻止系统进入集合 $P–Q$ 中的状态是安全机制的作用。实施这些限制的是安全机制。

访问控制矩阵模型是用于描述保护状态的有效模型，它准确地描述了一个主体（活跃的实体，如一个进程）对系统中其他实体的访问权限。访问控制矩阵 $\boldsymbol{A}$ 中的元素构成了一个当前系统状态的规范。规范有多种形式，现在研究者也提出了很多种描述规范的语言来描述系统中的允许状态。

保护状态随着系统的变化而变化。当一个系统命令改变了系统状态时，保护状态也随之转换。首先定义好一个允许状态的集合，在这个集合上有相应的允许操作。系统状态从一个允许状态开始，执行一个在该状态上的允许操作之后，系统状态发生了转换，转换的结果应该仍然是一个允许状态。如此递推下去，系统应该总是处于允许状态中。因此，允许状态和其他状态之间的转换是有限制的。

在实际的系统中，任何一个操作都会导致多个状态的转换，如读取数据、加载程序、修改数据和任何指令的执行都会导致系统状态的转换。只有系统中影响保护状态的状态转换才是值得关心的，也只有那些改变系统中实体被允许行为的状态转换才是我们感兴趣的。举例来说，如果一个程序将某个变量的值修改为 0，这种操作一般不会改变系统的保护状态。但是，如果这个变量会影响某个进程的优先级的话，程序做的这个修改就会影响到保护状态，此时就需要考虑状态转换了。

1.6.5　访问控制结构与设计原则

1975 年，邵泽（J.H. Saltzer）和施罗德（M.D. Schroeder）以安全保护机制的体系结构为中心，探讨了计算机系统的安全保护问题，重点考察了权能（Capability）和访问控制表（Access Control List）这两种结构，给出了设计安全保护机制的八大原则。

为讨论信息保护问题，从概念上可以为每个须保护的客体建立一个不可攻破的保护墙，保护墙上留有一个门，门前有一个卫兵，所有对客体的访问都首先在门前接受卫兵的检查。在整个系统中，有很多客体，因而有很多保护墙和卫兵。

对客体的访问控制机制的实现结构可分为两种类型：面向门票（Ticket-Oriented）的实现和面向名单（List-Oriented）的实现。

在面向门票的实现中，卫兵手中持有一份对一个客体的描述，在访问活动中，主体携带一张门票，门票上有一个客体的标识和可访问的方式，卫兵把主体所持门票中的客体标识与自己手中的客体标识进行对比，以确定是否允许访问；在整个系统中，一个主体可能持有多张门票。

在面向名单的实现中，卫兵手中持有一份所有授权主体的名单及相应的访问方式，在访问活动中，主体出示自己的身份标识，卫兵在名单中查找，检查主体是否记录在名单上，以确定是否允许访问。

权能结构属于面向门票的结构，一张门票也称做一个权能。访问控制表（ACL）结构属于面向名单的结构。在访问控制矩阵的概念模式下，权能结构对应访问控制矩阵的行结构，行中的每个矩阵元素对应一个权能；ACL 结构对应访问控制矩阵中的列结构，每一列对应一个 ACL。权能与访问控制表如图 1.7 所示。

邵泽和施罗德给出的设计安全保护机制的八大原则如下。

（1）经济性原则（Economy of Mechanism）。安全保护机制应设计得尽可能地简单和短小。有些设计和实现错误可能产生意想不到的访问途径，而这些错误在常规使用中

是察觉不出的，难免需要进行软件逐行排查等工作，简单而短小的设计是这类工作成功的关键。

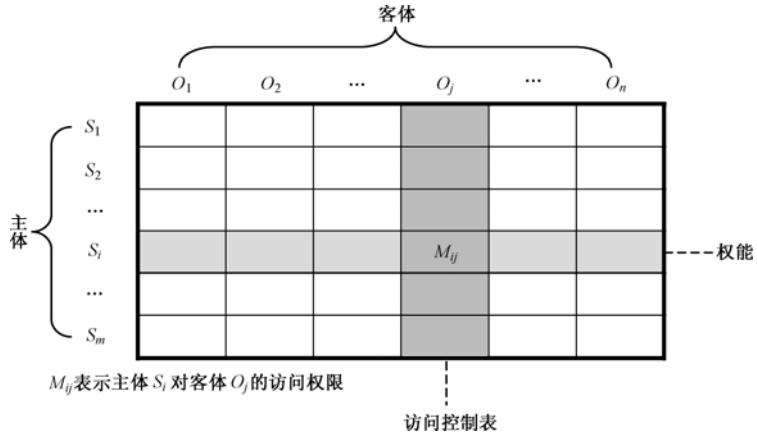


图 1.7　权能与访问控制表

（2）失败-保险原则（Fail-Safe Defaults）。访问判定应建立在显式授权而不是隐式授权的基础上，显式授权指定的是主体该有的权限，隐式授权指定的是主体不该有的权限。在默认情况下，没有明确授权的访问方式应该视为不允许的访问方式，如果主体欲以该方式进行访问，结果将是失败，这对于系统来说是保险的。

（3）完全仲裁原则（Complete Mediation）。对每一个客体的每一次访问都必须经过检查，以确认是否已经得到授权。

（4）开放性设计原则（Open Design）。不应该把保护机制的抗攻击能力建立在设计的保密性的基础之上。应该在设计公开的环境中设法增强保护机制的防御能力。

（5）特权分离原则（Separation of Privilege）。为一项特权划分出多个决定因素，仅当所有决定因素均具备时，才能行使该项特权。正如一个保险箱设有两把钥匙，由两个人掌管，仅当两个人都提供钥匙时，才能打开保险箱。

（6）最小特权原则（Least Privilege）。分配给系统中的每个程序和每个用户的特权应该是它们完成工作所必须享有的特权的最小集合。

（7）最少公共机制原则（Least Common Mechanism）。把由两个以上用户共用和被所有用户依赖的机制的数量减到最少。每一个共享机制都是一条潜在的用户间的信息通路，要谨慎设计，避免无意中破坏安全性。应证明为所有用户服务的机制能满足每个用户的要求。

（8）心理可接受性原则（Psychological Acceptability）。为使用户习以为常地、自动地正确运用保护机制，把用户界面设计得易于使用是根本。

邵泽和施罗德还指出：如何证明硬件和软件安全保护机制的设计与实现的正确性，是值得关注的重要研究课题。

1.7　系统安全知识定位

信息安全问题的解决涉及一个综合的系统工程。本教材中所说的信息系统安全指的是信息安全体系中的系统安全部分。不管一个信息系统的结构有多么复杂，其所承担的工作任务最终都要落实到具体的主机系统去完成，主机系统是信息系统的基础，所以，进一步地说，本教材将重点讲授主机系统的安全性问题。

1.7.1　系统安全的宏观定位

在认识事物的过程中，应该尽量避免出现“只见树木，不见森林”的局面，只有这样，才能更好地做到“知其然，亦知其所以然”。为了学好以主机为中心的信息系统安全性方面的知识，有必要从信息安全体系结构的整体安全需求的角度去了解系统安全的地位和作用。另一方面，当今的计算机系统基本上都与计算机网络密不可分，计算机网络是计算机系统工作的基本环境，以主机为中心的系统安全离不开网络安全，我们也应该从网络安全的角度去认识系统安全问题。系统安全的地位与环境如图 1.8 所示。

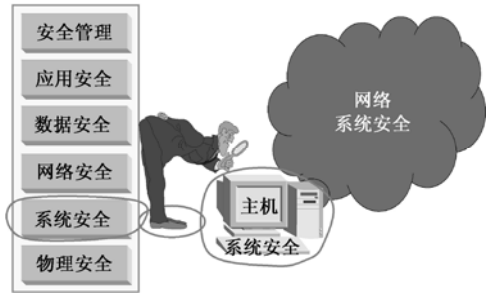


图 1.8　系统安全的地位与环境

图 1.8 的左侧给出的是整个信息安全体系结构中的整体安全需求。安全需求是信息安全体系结构设计与实施的源动力，涉及物理安全、系统安全、网络安全、数据安全、应用安全与安全管理等多个层面。

物理安全主要从外界环境、基础设施、运行硬件、介质等方面为信息安全提供基本的底层支撑和保障。实现物理安全的主要目的是确保存放计算机与网络设备的机房、信息系统设备和数据存储介质等免受物理环境、自然灾害及人为操作失误和恶意操作等各种威胁所产生的危害。

系统安全又称主机系统安全，主要提供安全的操作系统和安全的数据库管理系统，以实现操作系统和数据库管理系统的安全运行，包括服务器、终端/工作站等在内的计算机设备在操作系统和数据库系统层面的安全。系统安全是确保信息安全的中坚力量，是本课程要传授的主要知识。

网络安全为信息系统能够在安全的网络环境中运行提供支持。一方面，确保网络系统安全运行，提供有效的网络服务；另一方面，确保在网络上传输的数据的机密性、完整性和可用性等。

数据安全主要关注信息系统中存储、传输和处理等过程中的数据的安全性，其目的是实现数据的机密性、完整性、可用性和不可否认性等，并进行数据备份和恢复。

应用安全主要确保信息系统的各种业务的应用程序安全运行，其安全需求主要涉及身份认证机制和关键业务系统的对外接口，包括电子邮件、文件传输、语音通信、视频会议与视频点播、Web 网站等。

安全管理要落实的是信息系统在整个生命周期中涉及的安全管理制度、安全管理机构、人员安全管理、系统建设管理和系统运行维护管理等方面的安全需求。

1.7.2　系统安全的知识点定位

系统安全是以主机为中心的计算机系统与信息安全的交叉学科，系统安全的知识体系由计算机系统和信息安全中的相应知识体系融合而成。我们可以在计算机系统与信息安全这两个维度的结合点上考察系统安全中的基本问题，系统安全的二维知识体系如图 1.9 所示。

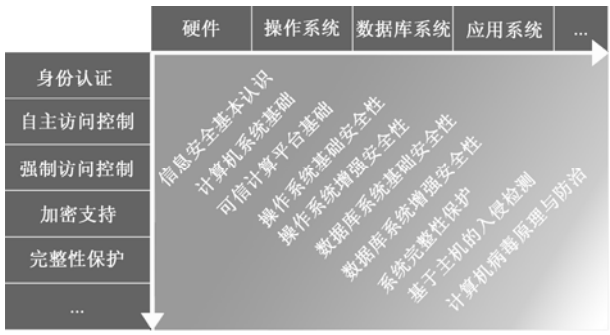


图 1.9　系统安全的二维知识体系

在计算机系统维度上，可以考查硬件、操作系统、数据库系统、应用系统等方面的内容。在信息安全维度上，可以考查身份认证、强制访问控制、加密支持、完整性保护等方面的内容。

根据计算机系统和信息安全这两个维度上的关键内容的融合特点，结合现实应用的实际需要，在本教材中，正如图 1.9 所示的那样，我们认为信息安全基本认识、计算机系统基础、可信计算平台基础、操作系统安全性、数据库系统安全性、系统完整性保护、基于主机的入侵检测及计算机病毒原理与防治等是在系统安全课程中应该把握的重要内容。

信息安全基本认识是本章要达到的主要目的，是系统安全课程的开端。计算机系统

基础将通过回顾计算机系统中硬件、操作系统、数据库系统等方面的基本内容来巩固，是建立系统安全思想的根基。可信计算平台通过安全芯片提供基本的安全功能，可以作为系统安全的硬件基础。

操作系统安全性和数据库系统安全性是系统安全的核心内容，值得分别从基础安全性和增强安全性两个层面去把握。系统完整性保护是系统安全核心内容的另一个重要方面，它不但可以体现在操作系统安全性和数据库系统安全性中，还可以体现在硬件的安全支持和应用系统的安全需求中。

操作系统和数据库系统在系统安全中处于核心地位，其核心意义主要体现在它们对于确保应用系统的安全性具有不可或缺的重要作用。应用系统的安全性是用户希望实现的根本目标。基于主机的入侵检测和计算机病毒原理与防治是系统安全由核心层向应用层延伸的重要内容。

1.8　本章小结

本章的设计思想是以信息安全攻击的实景为起点的，初步建立信息安全攻击威胁的感性认识，开始体验信息安全的基本理念，形成信息安全的基本认识，并透过信息安全的攻击威胁感受系统安全在信息安全中的地位和作用。本章主要由安全攻击实景呈现、安全攻击环节概览、信息安全典型事件、信息安全经典要素、信息系统安全策略、信息系统访问控制及系统安全知识定位等方面的内容构成。

由卡尔发动的旨在盗取信用卡信息的安全攻击事件展示了在开放信息网络空间中策划和实施信息安全攻击的基本过程和主要技术手段。卡尔的攻击具有获取经济利益的明确目的，反映了现代信息安全攻击的一种趋势。

信息网络空间中的安全攻击一般包含侦察、扫描、获取访问、维持访问和掩盖踪迹等主要环节。侦察就是收集有关被攻击目标的一般信息，扫描则是进一步掌握有关攻击目标的更多信息，获取访问就是闯入和访问目标系统，维持访问是在攻击成功之后为下一次攻击打开方便之门，掩盖踪迹指的是销毁攻击痕迹以便掩盖攻击行径。

机密性、完整性和可用性是信息安全的三个经典要素，这三个经典要素对应信息安全的三种基本安全服务。机密性是指对信息或资源的隐藏，完整性是指数据或资源的可信度，可用性是指对信息或资源的期望使用能力。

威胁是对安全的潜在破坏，这种破坏不一定要实际发生才成为威胁。威胁可以分成泄露、欺骗、破坏和篡夺四大类。泄露指对信息的非授权访问，欺骗指接受虚假数据，破坏指中断或妨碍正常操作，篡夺指对系统某些部分的非授权控制。

安全策略确定信息系统的安全性定义，安全策略与安全机制之间的差别对于安全性的研究非常重要，安全策略是有关允许什么和禁止什么的规定，安全机制是实施安全策略的方法、工具或规程。确定了描述“安全”和“非安全”行为的安全策略之后，安全

机制可以阻止攻击、检测攻击或在遭到攻击后进行恢复工作。

访问控制是确保信息系统安全的重要措施之一，访问控制矩阵是对访问控制行为的一种抽象表示，矩阵中的行与系统中的主体相对应，矩阵中的列与系统中的客体相对应，处于行与列的交叉点上的矩阵元素描述主体对客体的访问权限。

访问控制可以分为自主访问控制和强制访问控制两种类型。在自主访问控制中，用户可以参与确定访问控制的规则或属性；在强制访问控制中，只有特定的系统管理员才能确定访问控制的规则和属性。

贝尔–拉普杜拉模型是历史上第一个可证明的访问控制数学模型，它支持机密性强制访问控制。简单安全特性、星号安全特性、自主安全特性和基本安全定理是该模型的核心内容。该模型定义的安全级别是一个二元组，由等级分类和非等级类别两种元素构成，安全级别之间通过支配关系进行比较。

权能是面向门票的访问控制结构，一张门票对应一个权能。访问控制表是面向名单的访问控制结构。权能结构对应访问控制矩阵中的行结构，访问控制表结构对应访问控制矩阵中的列结构。

邵泽和施罗德于 1975 年给出的设计信息系统的八大原则包含经济性原则、失败-保险原则、完全仲裁原则、开放性设计原则、特权分离原则、最小特权原则、最少公共机制原则和心理可接受性原则。

为了学好以主机为中心的信息系统安全性方面的知识，有必要从信息安全体系结构的整体安全需求的角度去了解系统安全的地位和作用。以主机为中心的系统安全离不开网络安全，我们也应该从网络安全的角度去认识系统安全问题。



习题 1

1. 由卡尔发动的以盗取信用卡信息为目的的安全攻击可以划分为哪些阶段？在各个阶段中，卡尔主要做了哪些与攻击相关的事情？使用了哪些工具？

2. 开放网络互连环境中的信息安全攻击一般包含哪些主要环节？在各个环节中，攻击者主要想实现哪些目标？哪些环节与系统安全密切相关？为什么？

3. 历史上第一次大规模网络安全攻击事件发生在_____年，称为_____事件，关于该事件的特点，以下哪种说法是正确的？

A. 攻击方法复杂，但造成损失不大　　B. 攻击方法复杂，且造成损失巨大

C. 攻击方法简单，但造成损失巨大　　D. 攻击方法简单，且造成损失不大

4. 根据总部设在卡内基 • 梅隆大学的美国 CERT 提供的计算机安全事件数的统计数据，1988—2003 年，计算机安全事件数总体上呈什么样的变化态势？年均变化率是多少？

5. 试举例说明什么是信息安全中的机密性、完整性和可用性。

6. 信息安全威胁可以划分为哪几种主要类型？它们分别有什么含义？什么样的行为有可能产生这些威胁（为每种威胁举出一种行为）？

7. 什么样的系统可以称得上是一个安全的系统？如果一个系统是安全的，那么，这是否意味着该系统在任何场合都是安全的？为什么？

8. 什么是安全策略？什么是安全机制？安全策略与安全机制之间存在怎样的关系？试举例说明。

9. 安全策略有哪些常见类型？它们分别侧重信息安全的哪些经典要素？试举例说明每类安全策略的用途。

10. 什么是访问控制矩阵？试说明它的结构和含义，并举例说明它在实际应用中的应用方法。

11. 什么是自主访问控制？什么是强制访问控制？试分别举例加以说明。

12. 贝尔–拉普杜拉模型属于自主访问控制模型还是属于强制访问控制模型？为什么？

13. 设整数 1、2、3、4 分别表示非密、秘密、机密、绝密四种密级，符号 d_1、d_2、d_3 分别是财务处、科研处、教务处的标识，标签 l_1、l_2、l_3、l_4、l_5 用于表示贝尔–拉普杜拉模型中的安全级别，它们的定义如下：

$l_1 = (2, \{d_1\})$；$l_2 = (4, \{d_1\})$；$l_3 = (4, \{d_2\})$；$l_4 = (1, \{d_1, d_2, d_3\})$；$l_5 = (4, \{d_1, d_2, d_3\})$

试比较安全标签 l_1、l_2、l_3、l_4、l_5 之间的关系（支配关系）。

14. 根据第 13 题的定义，已知某学校下发的文件 F_1、F_2、F_3 的安全级别分别为 l_1、l_3、l_5，该校教师 T_1、T_2、T_3 的当前安全级别分别为 l_3、l_4、l_5，根据贝尔–拉普杜拉模型，试问：

（1）哪些教师可以读哪些文件？

（2）哪些教师可以写哪些文件？

15. 什么是权能结构？什么是访问控制表结构？试说明它们与访问控制矩阵之间的关系。

16. 由邵泽和施罗德于 1975 年提出的设计信息安全系统的八大原则包含的是哪些原则？它们分别有什么含义？

第 2 章
计算机系统基础



计算机系统由硬件和软件组成。硬件是计算机的物理设备。软件则是使硬件能够正常工作的程序的集合。计算机的硬件组成部件可以分为三大类（即三大子系统）：中央处理单元、主存储器和输入/输出子系统。基础的软件系统包括操作系统和数据库管理系统。计算机系统是系统安全的载体，为了更好地从整个计算机系统的角度讨论安全问题，本章简要回顾计算机硬件、操作系统和数据库系统等方面的基础知识。首先，我们以一个程序员的身份去领略一下计算机系统的基本面貌，然后，再通过“循规蹈矩”的方式去品味计算机组成、操作系统和数据库系统等方面的精髓。

2.1　程序员眼中的计算机系统

计算机系统是由硬件和系统软件组成的，它们共同工作来运行应用程序。虽然系统的具体实现方式随着时间不断变化，但是系统内在的概念却没有改变。所有计算机系统都由相似的硬件和软件组成，它们又执行着相似的功能，特别是在从程序员的角度看计算机系统时，情况更是如此。

柯尼汉（B. Kernighan）和里奇（D. Ritchie）在他们的 C 语言程序设计的经典著作中，设计了一个著名的 hello 程序，并用它来作为介绍 C 语言的典型样例，我们就借用它来帮助了解计算机系统，该程序如例 2.1 所示。

例 2.1　柯尼汉和里奇设计的 C 语言 hello 程序的源代码（hello.c）如下所示（行号是为了帮助说明问题而添加的）。

```
1.    # include <stdio.h>
2.
3.    int main()
4.    {
5.          printf(“hello, world\n”);
6.    }
```

尽管 hello 程序是一个非常简单的程序，但是为了完成它的执行，系统的每个主要组成部分都要协调工作。了解计算机系统的结构和工作原理，可以从了解 hello 程序的运行过程开始，我们需要弄清楚：当我们在系统上执行 hello 程序时，系统发生了什么及为什么会如此运作。我们通过跟踪 hello 程序的运行过程来学习计算机系统的基本知识。

假设已经把 hello.c 源程序编译成了可执行目标文件 hello，并存放在磁盘上。在 UNIX 系统中，可以按照例 2-1-2 所示的操作方法运行该可执行文件，所要做的事情是将它的文件名输入到被称为 shell 的应用程序中。

例 2.2　在 UNIX 系统的 shell 环境中，可以通过以下方式运行 hello 程序（行号是为了帮助说明问题而添加的）。

```
1.  unix>./hello
2.  hello, world
3.  unix>
```

第 1 行表示用户在 shell 的提示符下输入执行 hello 程序的命令，第 2 行表示该程序执行时输出的信息，第 3 行表示该程序执行结束。

shell 是一种命令行解释器，它输出一个提示符，等待用户输入一行命令，然后执行这个命令。如果该命令行的第一个单词不是一个内置的 shell 命令，那么 shell 就会假设这是一个可执行文件的名字，要加载和执行该文件。在我们的例子中，shell 将加载和执行 hello 程序，然后等待程序终止。hello 程序在屏幕上输出它的信息，然后终止。shell 随后输出一个提示符，等待下一个输入的命令行。

2.1.1　计算机系统的硬件组成

为了了解运行 hello 程序时发生了什么，我们需要了解一个典型的计算机系统的硬件组成，如图 2.1 所示。该图是 Intel Pentium 系统产品族的模型，但是所有其他系统也有相同的外观和特性。

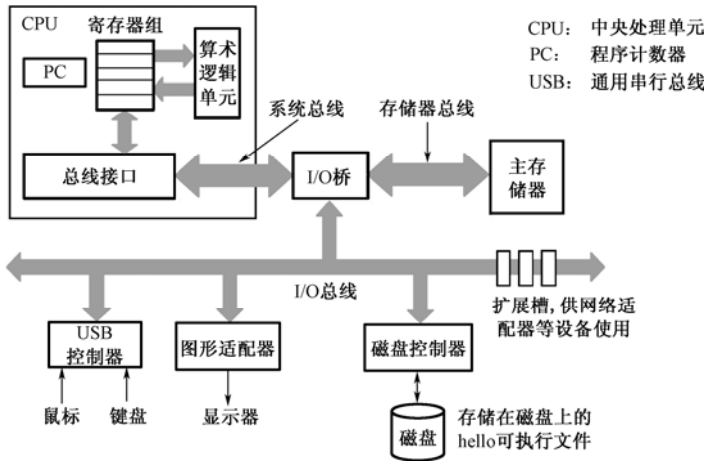


图 2.1　一个典型的计算机系统的硬件组成

根据图 2.1 容易看出，一个计算机系统由总线、I/O 设备、主存和处理器等硬件组件组成。

1．总线

贯穿整个系统的是一组电子线路，称做总线，它携带信息字节并负责在各个部件间传递。总线通常被设计成传送定长的字节块，也就是字（Word）。字中的字节数（即字长）是一个基本的系统参数，各个系统不尽相同。例如，Intel Pentium 系统的字长为 4 字节，而服务器类的系统，如 Intel Itaniums 和高端的 Sun 公司的 SPARCS 的字长为 8 字

节。用于汽车等工业中的嵌入式控制器之类的较小的系统的字长往往只有 1 或 2 字节。为了便于描述，我们假设字长为 4 字节，并且假设总线一次只传送 1 个字。

2．I/O设备

I/O（输入/输出）设备是系统与外界的联系通道。我们的示例系统包括 4 个 I/O 设备：作为用户输入的键盘和鼠标、作为用户输出的显示器及用于长期存储数据和程序的磁盘驱动器（简单地说就是磁盘）。最开始，可执行程序 hello 就放在磁盘上。

每个 I/O 设备都是通过一个控制器或适配器与 I/O 总线连接起来的。控制器和适配器之间的区别主要在于它们的组成方式。控制器是 I/O 设备本身或是系统的主印制电路板（通常被称做主板）上的芯片组，而适配器则是一块插在主板插槽上的卡。无论如何，它们的功能都是在 I/O 总线和 I/O 设备之间传递信息。

3．主存

主存是一个临时存储设备，在处理器执行程序时，用于存放程序和程序处理的数据。从物理上说，主存是由一组 DRAM（动态随机访问存储器）芯片组成的。从逻辑上说，存储器是由一个线性的字节数组构成的，每个字节都有自己唯一的地址（数组索引），这些地址是从零开始的。一般来说，组成程序的每条机器指令都由不定量的字节构成。与 C 程序变量相对应的数据项的大小是根据类型变化的。例如，在运行 Linux 的 Intel 机器上，short 类型的数据需要 2 个字节，int、float 和 long 类型则需要 4 个字节，而 double 类型则需要 8 个字节。

4．处理器

中央处理单元（CPU）简称处理器，是解释（或执行）存储在主存中的指令的引擎。处理器的核心是一个称为程序计数器（PC）的字长大小的存储设备，即寄存器（Register）。在任何一个时间点上，PC 都指向主存中的某条机器语言指令（内含其地址）。

从系统通电开始，直到系统断电，处理器一直在不假思索地重复执行相同的基本任务：从程序计数器（PC）指向的存储器处读取指令，解释指令中的位，执行指令指示的简单操作（Operation），然后更新程序计数器，使它指向下一条指令，而这条指令并不一定和存储器中刚刚执行的指令相邻。

这样的简单操作的数目并不多，它们在主存、寄存器组（Register File）和算术逻辑单元（ALU）之间循环。寄存器组是一个小的存储设备，由一些字长大小的寄存器组成，这些寄存器每个都有唯一的名字。ALU 计算新的数据和地址值。下面是一些简单操作的例子，CPU 在指令的要求下可能会执行这些操作。

（1）加载：从主存复制一个字节或一个字到寄存器，覆盖寄存器原来的内容。

（2）存储：从寄存器复制一个字节或一个字到主存的某个位置，覆盖这个位置上原来的内容。

（3）更新：复制两个寄存器的内容到 ALU，ALU 将两个字相加，并将结果存放到一个寄存器中，覆盖该寄存器中原来的内容。

（4）I/O 读：从一个 I/O 设备中复制一个字节或一个字到一个寄存器。

（5）I/O 写：从一个寄存器中复制一个字节或一个字到一个 I/O 设备。

（6）跳转：从指令本身抽取一个字，并将这个字复制到程序计数器（PC）中，覆盖 PC 中原来的值。

2.1.2　执行hello程序

通过对系统的硬件组成和操作的简单学习，我们可以开始了解运行示例程序时发生了什么。在这里必须忽略很多细节，稍后会做一些补充，但是现在我们将很满意于这种粗略的描述。

首先，shell 程序执行它的指令，等待用户输入命令。当用户通过键盘输入字符串“./hello”后，shell 程序就逐一读取字符到寄存器，再把它存放到存储器中。从键盘上读取 hello 命令如图 2.2 所示。

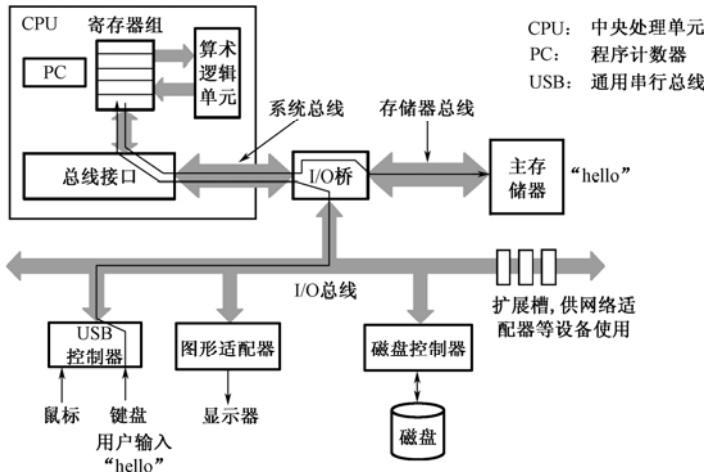


图 2.2　从键盘上读取 hello 命令

在键盘上按下回车键时，shell 就知道已经结束了命令的输入。然后 shell 执行一系列指令，这些指令将 hello 目标文件中的代码和数据从磁盘复制到主存，从而加载 hello 文件。数据包括最终会被输出的字符串“hello，world\n”。

利用称为 DMA（直接存储器访问）的技术，数据可以不通过处理器而直接从磁盘到达主存。从磁盘加载可执行文件到主存如图 2.3 所示。

一旦 hello 目标文件中的代码和数据被加载到了存储器，处理器就开始执行 hello 程序的主程序中的机器语言指令。这些指令将“hello，world\n”字符串中的字节从存储器中复制到寄存器组，再从寄存器中把文件复制到显示设备，最终显示在屏幕上。从存储

器写输出串到显示器如图 2.4 所示。

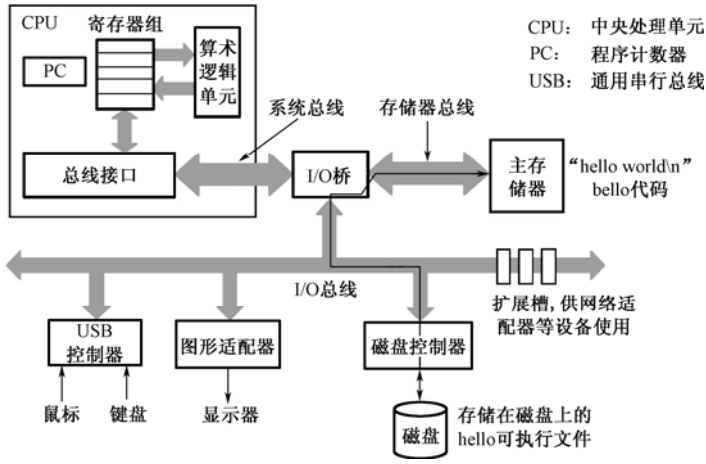


图 2.3　从磁盘加载可执行文件到主存

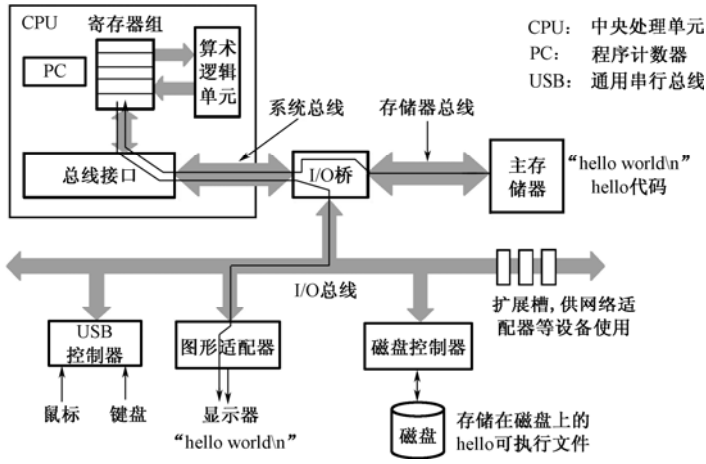


图 2.4　从存储器写输出串到显示器

2.1.3　高速缓存

通过这个简单的示例我们了解到重要的一点，那就是系统花费了大量的时间把信息从一个地方挪到另一个地方。hello 程序的机器指令最初是存放在磁盘上的；当程序加载时，它被复制到主存；当处理器运行程序时，指令又从主存复制到处理器。相似地，数据串“hello，world\n”开始时在磁盘上，接着被复制到主存，然后从主存复制到显示设备。从一个程序员的角度来看，大量的复制减慢了程序的实际工作速度。因此，系统设计者的一个主要目标就是使这些复制操作尽可能地快。

根据机械原理，较大的存储设备要比较小的存储设备运行得慢，而快速设备的造价远高于低速同类设备。例如，一个典型系统上的磁盘驱动器可能比主存大 100 倍，但是

对处理器而言，从磁盘驱动器上读取一个字的时间开销要比从主存中读取的开销大 1000 万倍。

类似地，一个典型的寄存器组只存储几百字节的信息，与此相反，主存里可存放几百万字节。然而，处理器从寄存器组中读取数据比从主存中读取要快几乎 100 倍。更麻烦的是，随着这些年半导体技术的进步，这种处理器与主存之间的速度差距（Processor Memory Gap）还在持续增大。加快处理器的运行速度比加快主存的处理速度要容易和便宜得多。

针对这种处理器与主存之间的差异，系统设计者采用了更小更快的存储设备，称为高速缓冲存储器（Cache Memory，简称高速缓存），它们被用来作为暂时的集结区域，存放处理器在不久的将来可能需要的信息。图 2.5 展示了一个典型系统中的高速缓冲存储器。位于处理器芯片上的 L1 高速缓存的容量可以达到数万字节，访问速度几乎和访问寄存器组一样快。一个容量为数十万到数百万的更大的 L2 高速缓存是通过一条特殊的总线连接到处理器的。进程访问 L2 的时间开销要比访问 L1 的开销大 5 倍，但是这仍然比访问主存的时间快 5～10 倍。L1 和 L2 高速缓存是用一种叫做静态随机访问存储器（SRAM）的硬件技术实现的。

应用程序开发人员通过理解高速缓冲存储器的机理，能够利用这些知识极大地提高程序的性能。

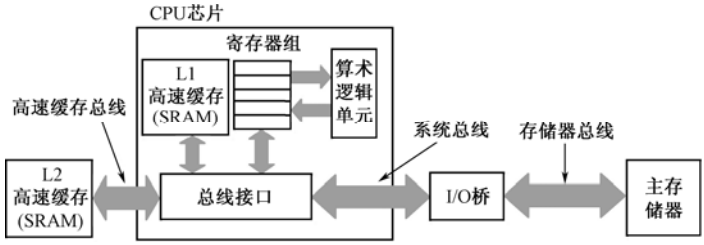


图 2.5　高速缓冲存储器

2.1.4　层次结构的存储设备

在处理器和一个较大、较慢的设备（如主存储器）之间插入一个较小、较快的存储设备（如高速缓冲存储器）的想法成为一个普遍的观念。实际上，每个计算机系统中的存储设备都被组织成一个存储器层次结构（Memory Hierarchy），就像图 2.6 所展示的那样。在这个层次结构中，从上至下，设备变得更慢、更大，并且每字节的造价也更便宜。寄存器组在层次结构中位于最顶部，也就是第 0 级或记为 L0；L1 高速缓存处在第一层（所以称为 L1）；L2 高速缓存占据第二层；主存在第三层；以此类推。

存储器分层结构的主要思想是某个层次上的存储器可作为下一层次上的存储器的高速缓存。因此，寄存器组可作为 L1 的高速缓存，而 L1 又可作为 L2 的高速缓存，L2 可作为主存的高速缓存，主存可作为磁盘的高速缓存。在某些带分布式文件系统的网络系

统中，本地磁盘可作为存储在其他系统中的磁盘上的数据的高速缓存。

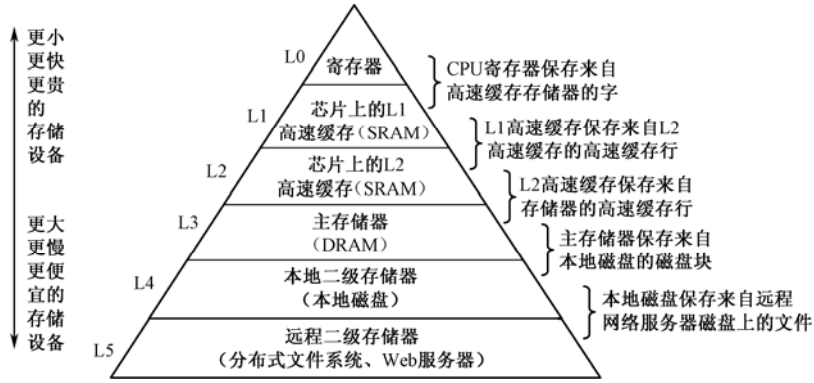


图 2.6　一个存储器层次结构的示例

就像程序员可以运用 L1 和 L2 的知识来提高程序的性能一样，程序员同样可以利用对整个存储器层次结构的理解来提高程序的性能。

2.1.5　操作系统管理硬件

让我们回到 hello 程序的例子。当 shell 加载和运行 hello 程序时，当 hello 程序输出自己的消息时，程序没有直接访问键盘、显示器、磁盘或主存储器。取而代之的是，它们依靠操作系统（Operating System）提供的服务。我们可以把操作系统看成是在应用程序和硬件之间插入的一层软件。计算机系统的分层视图如图 2.7 所示。所有应用程序对硬件的操作尝试都必须通过操作系统进行。

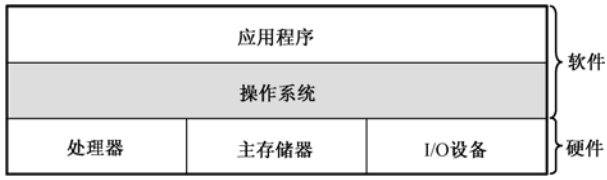


图 2.7　计算机系统的分层视图

操作系统有两个主要用途：一是防止硬件被失控的应用程序滥用；二是在控制复杂而通常又颇具差异的低级硬件设备方面，为应用程序提供简单一致的方法。操作系统通过如图 2.8 所示的几个基本的抽象概念（进程、虚拟存储器和文件）实现这两个用途。如图 2.8 所示，文件是对 I/O 设备的抽象表示，虚拟存储器是对主存储器和磁盘 I/O 设备的抽象表示，进程则是对处理器、主存储器和 I/O 设备的抽象表示。下面将依次讨论每种抽象表示。

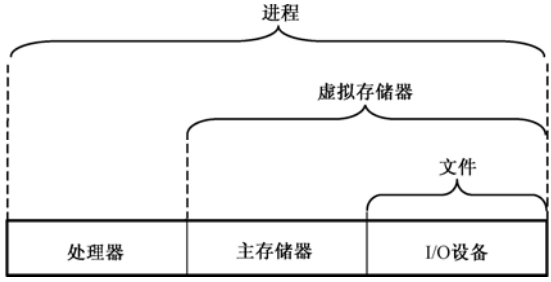


图 2.8　操作系统提供的抽象表示

1．进程

像 hello 这样的程序在现代计算机系统上运行时，操作系统会提供一种假象，就好像系统上只有这个程序在运行。程序看上去独占地使用处理器、主存储器（主存）和 I/O 设备，而处理器看上去就好像在不间断地一条接一条地执行该程序中的指令。该程序的代码和数据就好像是系统存储器中唯一的对象。这些假象是通过进程的概念来实现的，进程是计算机科学中最重要和最成功的概念之一。

进程是操作系统对运行的程序的一种抽象。在一个计算机系统上可以同时运行多个进程，而每个进程都好像在独占地使用硬件，我们称之为并发运行，实际上，一个进程的指令和另一个进程的指令是交错执行的。操作系统中实现这种交错执行功能的机制称为环境切换（Context Switching）。

操作系统保存进程运行所需的所有状态信息。这种状态，也就是环境（Context），包括许多信息，如 PC 和寄存器组的当前值及主存的内容。在任何一个时刻，系统上都只有一个进程正在运行。当操作系统决定把控制权从当前进程转移给某个新进程时，它就会进行环境切换，即保存当前进程的环境、恢复新进程的环境，然后将控制权转移给新进程。新进程就会从它上次停止的地方开始执行。图 2.9 展示了示例 hello 运行的基本场景中进程的环境切换。

实现进程这个抽象概念需要低级硬件和操作系统软件的紧密合作。

进程这个抽象概念还暗示着由于不同的进程交错执行，打乱了时间的概念，使得程序员很难获得运行时间的准确和可重复测量。现代计算机系统建立了各种时间概念，并支持用来获得准确测量值的技术。

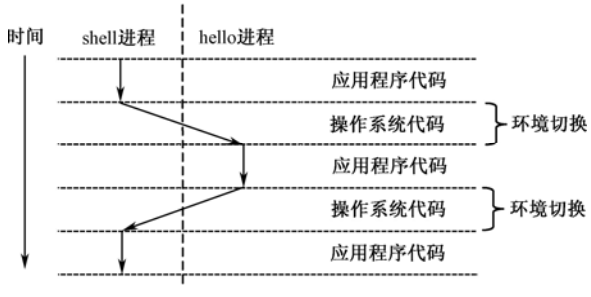


图 2.9　进程的环境切换

2．线程

尽管通常认为一个进程只有单一的控制流，但是在现代计算机系统中，一个进程实际上可以由多个称为线程的执行单元组成，每个线程都运行在进程的环境中，并共享同样的代码和全局数据。由于网络服务器中对并行处理的要求，线程成为越来越重要的编程模型，因为多线程之间比多进程之间更容易共享数据，也因为线程一般情况下都比进程更高效。

3．虚拟存储器

虚拟存储器是一个抽象概念，它为每个进程提供了一个假象，好像每个进程都在独占地使用主存。每个进程看到的存储器都是一致的，称之为虚拟地址空间。图 2.10 所示的是 Linux 进程的虚拟地址空间（其他 UNIX 系统的设计也与此类似）。在 Linux 系统中，最上面的四分之一的地址空间是预留给操作系统中的代码和数据的，这对所有进程都一样。底部的四分之三的地址空间用来存放用户进程定义的代码和数据。请注意：图中的地址是从下往上增大的。

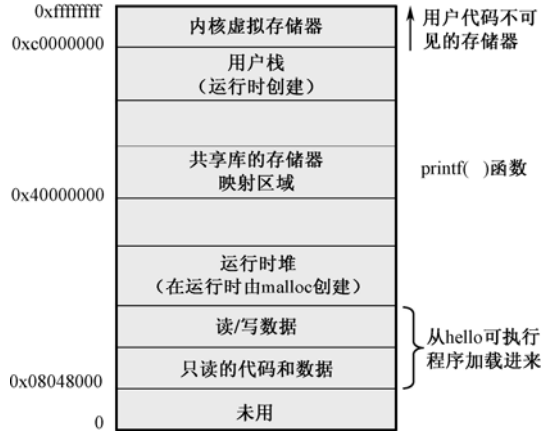


图 2.10　Linux 进程的虚拟地址空间

每个进程看到的虚拟地址空间由大量准确定义的区（Area）构成，每个区都有专门的功能。让我们简单地看一看每个区，从最低的地址开始，逐步向上研究将是非常有益的。

（1）程序代码和数据：代码是从同一固定地址开始的，紧接着的是和 C 全局变量相对应的数据区。代码和数据区是由可执行目标文件直接初始化的，在我们的示例中就是可执行文件 hello。

（2）堆：代码和数据区后紧随着的是运行时堆。代码和数据区是在进程开始运行时就被指定了大小的，与此不同，作为调用像 malloc 和 free 这样的 C 标准库函数的结果，堆可以在运行时动态地扩展和收缩。虚拟存储器管理对堆有详细的研究。

（3）共享库：在地址空间的中间附近是一块用来存放像 C 标准库和数学库这样的共

享库的代码和数据的区域。共享库的概念非常强大，但是也是个相当难懂的概念。共享库借助动态链接技术进行工作。

（4）栈：位于用户虚拟地址空间顶部的是用户栈，编译器用它来实现函数调用。和堆一样，用户栈在程序执行期间可以动态地扩展和收缩。特别地，每调用一个函数，栈就会增长；每次从函数返回时，栈就会收缩。编译器确定栈的使用方法。

（5）内核虚拟存储器：内核是操作系统中总是驻留在存储器中的部分。地址空间顶部的四分之一部分是为内核预留的。应用程序不允许读/写这个区域的内容或直接调用内核代码定义的函数。

虚拟存储器的运作需要硬件和操作系统软件间的精密复杂的互相合作，包括对处理器生成的每个地址的硬件翻译。基本思想是把一个进程虚拟存储器的内容存储在磁盘上，然后把主存作为磁盘的高速缓存。

4．文件

文件就是字节序列。每个 I/O 设备，包括磁盘、键盘、显示器，甚至网络，都可以被看成是文件。在 UNIX 系统中，所有的输入/输出都是通过使用称为 UNIX I/O 的一小组系统调用函数读/写文件来实现的。

文件这个简单而精致的概念是非常强大的，因为它使得应用程序能够统一地看待系统中可能含有的所有各式各样的 I/O 设备。例如，处理磁盘文件内容的应用程序员无须了解具体的磁盘技术。进一步说，同一个程序可以在使用不同磁盘技术的不同系统上运行。

2.1.6　通过网络与其他系统通信

在前面的系统漫游过程中，一直把计算机系统视为一个孤立的硬件和软件的集合体。实际上，现代计算机系统经常是通过网络和其他系统连接在一起的。从一个单独的系统来看，网络可被视为又一个 I/O 设备，如图 2.11 所示。当系统从主存复制一串字符到网络适配器时，数据流经过网络到达另一台机器，而不是到达本地磁盘驱动器。相似地，系统可以读取从其他机器发送来的数据，并把数据复制到自己的主存。

随着像因特网这样的全球网络的出现，从一台主机复制信息到另外一台主机已经成为计算机系统最重要的用途之一。例如，像电子邮件、即时消息传送、万维网、FTP 和 telnet 这样的应用都是基于通过网络复制信息的功能的。

回到 hello 示例，可以使用熟悉的 telnet 应用在一个远程主机上运行 hello 程序。假设用本地主机上的 telnet 客户端连接远程主机上的 telnet 服务器。在登录到远程主机并运行 shell 后，远端的 shell 就在等待接收输入的命令。从这点上来看，在远端运行 hello 程序包括如图 2.12 所示的五个基本步骤。

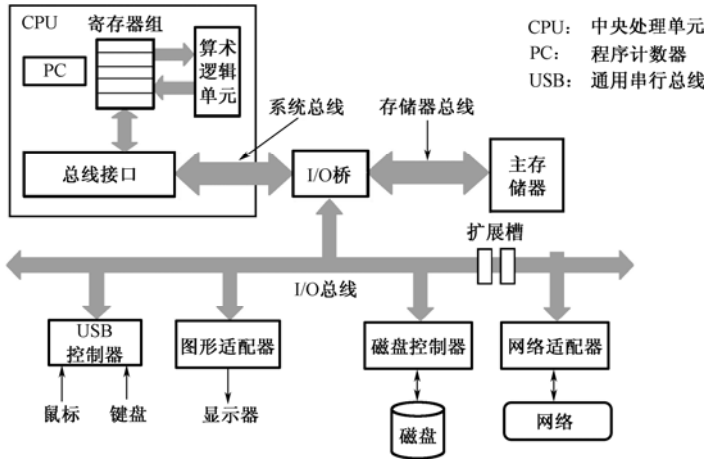


图 2.11　网络也是一种 I/O 设备

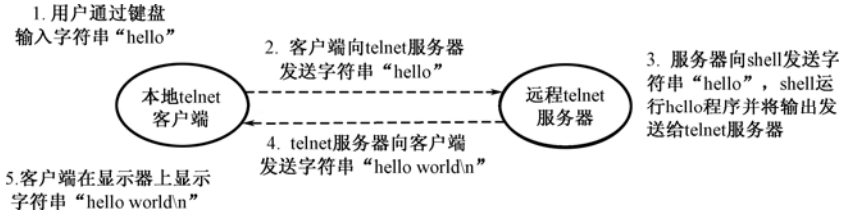


图 2.12　利用 telnet 跨越网络远程运行 hello 程序

当在 telnet 客户端输入“hello”字符串并按下回车键后，客户端软件就会将这个字符串发送到 telnet 的服务器。telnet 服务器从网络上接收到这个字符串之后，会把它传递给远端的 shell 程序。接下来，远端的 shell 运行 hello 程序，并将输出行返回给 telnet 服务器。最后，telnet 服务器通过网络把输出串转发给 telnet 客户端，客户端就将输出串输出到本地终端上。

这种在客户端和服务器之间交互的类型在所有的网络应用中都是非常典型的。

2.2　计算机组成基础

2.1 节从程序员的视角对计算机系统的基本构成和工作原理进行了探讨。本节从计算机系统的自身特点出发，了解计算机系统的基本组成和基本功能，主要介绍中央处理单元、主存储器和输入/输出子系统，以及各子系统相互沟通的原理。

2.2.1　中央处理单元

中央处理单元（CPU）用于数据的运算。它有三个组成部分：算术逻辑单元（ALU）、寄存器、控制单元。

1．算术逻辑单元

算术逻辑单元（ALU）用于进行算术运算和逻辑运算。

最简单的一元算术运算是增量（加 1）和减量（减 1）运算。最简单的二元算术运算是加、减、乘和除。控制单元负责选择这些运算中的一种。

最简单的一元逻辑运算是非，最简单的二元逻辑运算是与、或、异或。控制单元负责选择这些运算中的一种。

2．寄存器

寄存器是用来临时存放数据的高速独立的存储单元。CPU 的运算离不开多个寄存器。

1）数据寄存器

过去，计算机只有一个寄存器用来交替存储输入数据（直接来自内存的其他输入数据）或结果。现在，越来越多的复杂运算改由硬件设备实现（而不是使用软件），并且需要一些寄存器来保存这些运算的中间结果，这也就是为什么在计算机 CPU 内部使用大量寄存器来提高运算速度的原因。

2）指令寄存器

现在，计算机存储的不只是数据，还存储内存中相对应的程序。CPU 的主要职责是：从内存中逐条地取出指令，并将取出的指令存储在指令寄存器中，解释并执行指令。

3）程序计数器

CPU 中另一个通用寄存器是程序计数器。程序计数器保存当前正在执行的指令。当前的指令执行完后，计数器将自动加 1，指向下一条指令的内存地址。

3．控制单元

CPU 的第 3 个部分是控制单元，控制单元类似于人脑中控制身体各部分运动的区域。控制是通过线路的开（高电平）或关（低电平）来实现的。例如，简单的 ALU 可能要执行 10 种不同的运算，为了指定这些运算，控制单元需要通过 4 条线路连接到算术逻辑单元，4 条线路可以表示 16 种不同的状态（2^4），其中可以使用 10 种来进行算术和逻辑运算，其他的可作别的用途。可以用 0 表示线路断开，1 表示线路接通。这样，线路的状态就可以定义为 0000、0001、0010、…、1111。可以定义 0000（所有的线路断开）表示不进行任何运算，0001 表示加，0010 表示减，等等。

2.2.2　主存储器

主存储器是计算机内的另一个子系统。它是存储单元的集合，每一个存储单元都有唯一的标识符，称为地址。数据以称之为字的位组的形式在存储器中传入和传出。字可以是 8 位、16 位、32 位，甚至有时是 64 位，如果字是 8 位的话，一般称之为字节。术语字节在计算机科学中使用相当普遍，因此有时称 16 位为两个字节，称 32 位为 4 个

字节。

1．地址空间

在存储器中存取每个字都需要有相应的标识符。尽管程序员使用命名的方式来区分字（或一组字的集合），但在硬件层次上，每个字都是通过地址来标识的。所有在存储器中可标识的独立地址单元的总数称为地址空间。例如，一个 64K 字节、字长为 1 字节的存储器的地址空间的范围为 0～65 535。

由于计算机都是以位模式存储数据并进行运算的，因此地址本身也是用位模式表示的。如果内存是 64KB（2^{16}），字长为一个字节，那么就需要 16 位的位模式来确定地址。地址可以用无符号整数表示（不用负的地址）。或者说，起始地址通常是 0000000000000000（地址 0），最后一个地址通常是 1111111111111111（地址 65535）。通常，如果计算机有 N 个字的存储空间，那就需要有 $\log_2 N$ 位的无符号整数来确定每一个存储单元。

存储地址用无符号二进制整数定义。

例 2.3　一台计算机有 32MB（兆字节）主存（内存）。需要多少位来寻址内存中的任意一个字节？

解答：内存地址空间是 32MB，即 2^{25}（$2^5 \times 2^{20}$）。这就意味着需要 $\log_2 2^{25}$，即 25 位来标识每一个字节。

例 2.4　一台计算机有 l28MB 内存。计算机字长为 8 字节，需要多少位来寻址内存中的任意一个单字？

解答：内存地址空间是 l28MB，就是 2^{27}。但是，每个字是 8（2^3）字节，那也就意味着需要 2^{24} 个字，也就是说要 $\log_2 2^{24}$，即 24 位来标识每一个单字。

2．存储器的种类

有两种类型的存储器：RAM 和 ROM。

1）RAM

随机存取存储器（RAM）是计算机中主存的主要组成部分。该术语有时因为 ROM 也能随机存取而与 ROM 混淆。RAM 和 ROM 的区别在于：用户可读写 RAM，即用户可以在 RAM 中写信息，之后可以方便地通过覆盖来擦除原有信息。RAM 的另一个特点是易失性。当系统断电后信息（程序或数据）将被擦除。换句话说，如果计算机断电或关掉计算机，储存在 RAM 中的信息将被删除。RAM 技术又可以分为两大类：SRAM 和 DRAM。

（1）SRAM。静态 RAM（SRAM）技术是用传统的触发器门电路（有 0 和 1 两个状态的门）来保存数据的。这些门保持状态（0 或 1），这也就是说当通电的时候数据始终存在，不需要刷新。SRAM 速度快但是价格高。

（2）DRAM。动态 RAM（DRAM）技术使用电容器。如果电容器充电，则这时的状态是 1；如果放电，则状态是 0。因为电容器会随时间而漏掉一部分电，所以内存单元需要周期性地刷新。DRAM 速度慢，但是便宜。

2）ROM

只读存储器（ROM）的内容是由制造商写入的。用户只能读但不能写，它的优点是非易失性，当切断电源后，数据也不会丢失。通常用来存储那些关机后也不能丢失的程序或数据。例如，用 ROM 来存储那些在开机时运行的引导程序。

PROM 是称为可编程只读存储器（PROM）的一种 ROM。这种存储器最初时是空白的。计算机用户借助一些特殊的设备可以将程序存储在上面。当程序存储后，它就会像 ROM 一样不能够重写。也就是说计算机用户可以用它来存储一些特定的程序。

EPROM 是称为可擦除的可编程只读存储器（EPROM）的一种 PROM。用户可以对它进行编程，但是得用一种可以发出紫外光的特殊仪器对其进行擦写。EPROM 存储器需要拆下来擦除再重新安装。

EEPROM 是称为电可擦除的可编程只读存储器（EEPROM）的一种 EPROM。对它的编程和擦除用电子脉冲即可，无须从计算机上拆下来。

3．存储器的层次结构

计算机用户需要许多存储器，尤其是速度快且价格低廉的存储器。但这种要求并不总能得到满足。存取速度快的存储器通常都不便宜。因此需要寻找一种折中的办法。解决的办法是采用存储器的层次结构。该层次结构如下。

（1）当对速度要求很苛刻时可以使用少量高速存储器。CPU 中的寄存器就是这种存储器。

（2）用适量的中速存储器来存储经常需要访问的数据。例如下面将要讨论的高速缓冲存储器，就属于这一类。

（3）用大量的低速存储器存储那些不经常访问的数据。主存就属于这一类。

4．高速缓冲存储器

高速缓冲存储器的存取速度要比主存快，但是比 CPU 和 CPU 内部的寄存器要慢。高速缓存通常容量较小，且常被置于 CPU 和主存之间。

高速缓冲存储器在任何时候都含有主存中一部分内容的副本。当 CPU 要存取主存中的字时，将按以下的步骤进行：

（1）CPU 首先检查高速缓存。

（2）如果要存取的字存在，CPU 就将它复制；如果不存在，CPU 将从主存中复制一份从需要读取的字开始的数据块。该数据块将覆盖高速缓存中的内容。

（3）CPU 存取高速缓冲存储器并复制这个字。

这种方式将提高运算的速度。如果字在高速缓存中，就立即存取它；如果字不在高

速缓存中，字或整个数据块就会被复制到高速缓存中。因为很有可能 CPU 在下次存取中需要存取这次存取的第一个字的后续字，所以高速缓存可以提高处理速度。

为什么尽管高速缓存存储容量小效率却很高？这是由于 80-20 规则。据观察，通常计算机花费 80%的时间来读取 20%的数据。换句话说，相同的数据往往被存取多次。高速缓冲存储器凭借其高速，可以存储这 20%的数据而使存取速度至少快 80%。

2.2.3　输入/输出子系统

计算机中的第三个子系统是称为输入/输出（I/O）子系统的一系列设备。这个子系统可以使计算机与外界通信，并在断电的情况下存储程序和数据。输入/输出设备可以分为两个大类：非存储设备和存储设备。

1．非存储设备

非存储设备使得 CPU/内存可以与外界通信，但它不能存储信息。

1）键盘和监视器

两个最常见的非存储输入/输出设备是键盘和监视器。键盘提供输入功能，监视器显示输出并同时响应键盘的输入。程序、命令和数据的输入或输出都是通过字符串进行的。字符则是通过字符集（如 ASCII 码）进行编码的。

2）打印机

打印机是一种用于产生永久记录的输出设备。打印机是非存储设备，因为打印出来的资料不能够直接再输入到计算机中，除非有人通过打字或扫描的方式把它输入到计算机中。

2．存储设备

尽管存储设备被分为输入/输出设备，但它可以存储大量的信息以备后用。它们要比主存便宜得多，而且它们存储的信息也不易丢失（即使断电信息也不会丢失）。有时称它们为辅助存储设备，通常分为磁介质存储设备和光介质存储设备两种。

1）磁介质存储设备

磁介质存储设备利用磁性来存储数据位。如果点有磁性则表示 1，如果点没有磁性则表示 0。

a. 磁盘

磁盘是由一张一张的磁片叠加而成的。这些磁片由薄磁膜封装起来。信息是通过盘上每一个磁片的读/写磁头读写磁介质表面来进行读取和存储的。

表面结构：为了将数据存储在磁盘的表面，每个盘面都被划分成磁道，每个磁道又被分成若干个扇区。磁道间通过磁道内部间隔隔开，扇区之间通过扇区内部间隔隔开。

数据存取：磁盘是随机存取设备。但是，在某一时间可以读取的最小存储区域只能是一个扇区。数据块可以存储在一个或多个扇区上，而且该信息的获取不需要读取磁盘

上的其他信息。

性能：磁盘的性能取决于几个因素。最重要的因素是角速度、寻道时间和传送时间。角速度定义了磁盘的旋转速度。寻道时间定义了读/写磁头寻找数据所在磁道的时间。传送时间定义了将数据从磁盘移到 CPU/内存所需要的时间。

b. 磁带

磁带大小不一。最普通的一种是用厚磁膜封装的半英寸塑料磁带。磁带用两个滚轮承接起来，当转动的磁带通过时，就可以通过读/写磁头来读写磁带上的数据。

表面结构：磁带的宽度可以分为 8 个磁道；磁道上的每个点可以存储 1 位的信息。垂直的 9 个点可以存储 8 个位（即一个字节）的信息，还有 1 位用于错误检测。

数据存取：磁带是顺序存取设备。尽管磁带的表面可能会分成若干块，但是却没有寻址机制来读取每个块。要想读取指定的块就需要按照顺序通过其前面所有的块。

性能：尽管磁带的速度比磁盘慢，但它非常便宜。现在，人们使用磁带来备份存储大量的数据。

2）光存储设备

光存储设备是一种新技术，它使用光（激光）技术来存储和读取数据。在发明了 CD（光盘）后，人们利用光存储技术来保存音频信息。现在，相同的技术（稍作改进）被用来存储计算机上的信息、使用这种技术的设备有只读光盘（CD-ROM）、可刻录光盘（CD-R）、可重写光盘（CD-RW）、数字多功能光盘（DVD）。

（1）CD-ROM。只读光盘（CD-ROM）使用与 CD（光盘）相同的技术（该技术最初是由飞利浦和索尼公司为录制音乐而研发的）。两者间的唯一区别在于增强程度，CD-ROM 更健壮，而且纠错能力较强。

（2）CD-R。CD-ROM 技术只有在生产商大批量生产时才合理。另一方面，可刻录光盘（CD-R）则可以让用户自己制作一张或更多的盘，而不必考虑像制作 CD-ROM 的一些开销。它非常适合做备份，用户只需要一次写入信息，就可以多次读出信息。这也是它为什么有时被称为“写一次，读多次”（WORM）的原因。

（3）CD-RW。尽管 CD-R 已很受欢迎，但它们只能被写一次。为了能重写以前的资料，便有了一项新的技术，利用该技术可以制作一种称为可重写光盘（CD-RW）的新型盘，有时也称之为可擦写光盘技术。

（4）DVD。工业界已经有了对更大存储容量的数字存储介质的需求。CD-ROM（650MB）的存储容量已经不能满足视频信息存储的需要。市场上最新的光存储设备叫做数字多功能光盘（DVD）。

2.2.4　子系统的内部连接

前面介绍了在单个计算机上的（CPU、主存和输入/输出）三个子系统的主要特点。

本节将介绍三者之间在内部是如何连接的，内部连接扮演着很重要的角色，因为信息需要在这三个子系统中交换。

1．CPU和主存的连接

CPU 和主存之间通常由称为总线的三组线路连接在一起，它们分别是：数据总线、地址总线和控制总线。

1）数据总线

数据总线是由多根线组成的，每根线上每次传送 1 个位的数据。线的数量取决于字的大小。例如，如果计算机的字是 32 位（4 个字节），那么需要有 32 根线的数据总线，以便同一时刻能够同时传送 32 位的字。

2）地址总线

地址总线允许访问主存中的某个字，地址总线的线数取决于存储空间的大小。如果存储容量为 2^n 个字，那么地址总线一次需要传送 n 位的地址数据。因此它需要 n 根线。

3）控制总线

控制总线负责在中央处理器和主存之间传送信息。例如，必须有一个代码从 CPU 发往主存，用于指定进行读操作还是写操作。控制总线的线数取决于计算机所需要的控制命令的总数。如果计算机有 2^m 条控制命令，那么控制总线就需要有 m 根，因为 m 位可以定义 2^m 个不同的操作。

2．I/O设备的连接

输入/输出设备不能直接与连接 CPU 和主存的总线相连。因为输入/输出设备的本质与 CPU 和主存的本质不同，输入/输出设备都是机电、磁性或光学设备，而 CPU 和主存是电子设备。与 CPU 和主存相比，输入/输出设备的操作速度要慢得多。因此必须要有中介来处理这种差异，输入/输出设备是通过一种被称为输入/输出控制器或接口的器件连接到总线上的。

控制器，或者说接口，清除了输入/输出设备与 CPU 及主存在本质上的障碍。控制器可以是串行或并行设备。串行控制器只有一根线连接到设备上，而并行控制器则有数根线连接到设备上，使得一次能同时传送多个位。

有几种控制器至今还在使用，如今最常用的有 SCSI、火线和 USB。

1）SCSI

小型计算机系统接口最初是 1984 年为 Macintosh 计算机而设计的。今天还有许多系统使用它。它是一个 8、16 或 32 线的并行接口。SCSI 接口提供了菊花链连接，连接链的两端都必须有终结器，并且每个设备都必须要有唯一的地址（目标 ID）。

2）火线

IEEE 标准 1394 规定的串行接口俗称为火线。它是一种高速串行接口，数据采用数据包的形式传送，数据的传输速度高达 50MB/s。它可以在一条菊花链或树型连接（只

用一根线）上连接多达 63 个设备。它不需要 SCSI 控制器中那样的终结器。

3）USB

通用串行总线（USB）控制器是一种可以和火线控制器相媲美的控制器。USB 也是一种串行控制器，用以连接与计算机相连的一些低速设备（如键盘、鼠标等）。它的数据传输速度可达 1.5MB/s。它有 4 线总线，其中 2 线是用来为设备提供电源的。

3．输入/输出设备的寻址

通常 CPU 使用相同的总线向主存和输入/输出设备读写数据。唯一的不同是指令。如果指令涉及主存中的字，那么数据会在主存和 CPU 之间传送，如果指令涉及输入/输出设备，那么数据会在输入/输出设备和 CPU 之间传送。有两种方法用来对输入/输出设备进行寻址，即 I/O 独立寻址和 I/O 存储器映射寻址这两种方式。

1）I/O独立寻址

在 I/O 独立寻址中，用来读/写主存的指令与用来读/写输入/输出的指令是完全不同的。有专门的指令完成对输入/输出设备的测试、控制及读写操作。每个输入/输出设备都有自己的地址。因为指令不同，所以输入/输出地址可以和主存地址交叠而不会产生混淆。例如，CPU 可以使用读命令 Read 05 来从主存中读取字 05，它也可以使用输入命令 Input 05 从地址端口为 05 的输入/输出设备中读取数据。这里不会发生混淆是因为 Read 指令是规定从主存中读取数据的，而 Input 指令则是规定从输入/输出设备中读取数据的。

2）I/O存储器映射寻址

在 I/O 存储器映射寻址方式中，CPU 将输入/输出控制器中的每个寄存器都看做是主存中的字。换言之，CPU 没有单独的指令用来表示是从主存或是从输入/输出设备传送数据。例如，在指令集中只有一条 Read 指令，如果地址指定的是主存中的某个字，则从主存中读取数据；如果地址指定的是输入/输出设备中的某个寄存器，那么就从寄存器中读取数据。主存映射的配置优点在于指令集较小，所有对主存的操作指令都同样适用于输入/输出设备，其缺点则是输入/输出控制器中的寄存器占用了一部分主存地址。例如，假如有 5 个输入/输出控制器，每个控制器有 4 个寄存器，则共占用 20 个地址。相应的主存的大小就减小了 20 个字。

2.3　操作系统基础

操作系统是计算机硬件和用户（程序和人）的接口，它使得其他程序更加方便、有效地执行，并能方便地对计算机硬件和软件资源进行访问。

操作系统的两个主要设计目标是：更有效地使用硬件；更容易地使用资源。

操作系统是非常复杂的系统，因此很难给予它一个普遍认同的定义。我们在这里列举一些通用的定义。

- 操作系统是介于计算机硬件和用户（程序或人）之间的接口。
- 操作系统是一种使得其他程序更加方便、有效执行的程序（或一组程序）。
- 操作系统作为通用管理程序管理着计算机系统中每个部件的活动，并确保计算机系统中的硬件和软件资源能够更加有效地使用。当资源使用出现冲突时，操作系统应能够及时处理、排除冲突。

2.3.1　操作系统的发展及其意义

自 1946 年 2 月 14 日世界上第一台电子计算机“埃尼阿克”（ENIAC）在美国宾夕法尼亚州立大学诞生以来，计算机技术的发展以惊人速度向前推进着。以计算机技术为核心的信息技术领域发生了翻天覆地的变化。

回首当年，“埃尼阿克”揭开了电子计算机发展和应用的序幕，宣告了一个时代的开始。从规模上看，“埃尼阿克”是个“笨重”的家伙，因为它长 30.48m、宽 1m、占地面积为 170m^2、重达 30 t、拥有 30 个操作台、相当于 10 间普通房间的大小，所以称得上是地地道道的庞然大物，计算机专家操作“埃尼阿克”的图片如图 2.13 所示。



图 2.13　计算机专家在操作“埃尼阿克”

说“埃尼阿克”笨，是因为它的智能太低，只有专家才会使用，普通用户只能望而却步。究其原因，就是它缺乏操作系统。

今天，计算机已经可以做得非常轻巧，并且早已进入寻常百姓家。“轻”，当然是硬件发展的卓越成果。“巧”，则得益于装备了功能丰富的操作系统。操作系统带来的易操作性使很多用户甚至根本意识不到自己其实是在使用计算机。试想，有多少用户在使用智能手机时会想到自己实际上是在操作一台功能比“埃尼阿克”提高了成千上万倍的计算机呢？

不只是“埃尼阿克”，早期的计算机都没有操作系统。在没有操作系统的年代，使用计算机并不是一件容易的事情。

起初，计算机用户只能通过操作一排排的机械开关，把程序一位一位地输入到计算机中。逐渐地，出现了机器语言，用户可以把用机器语言编写的程序以打孔的形式记录在穿孔卡或纸带上，让计算机从穿孔卡或纸带上读取程序。可见，仅是把程序输入到计算机中就是一件烦琐的工作。

同样，从计算机中获取输出结果也并不轻松。为了减轻用户使用计算机时的输入/输出负担，人们为计算机设计了输入/输出控制系统（IOCS，Input-Output Control System）。该系统的实现也许可以算得上是今天操作系统思想的开端。的确，计算机的人机交互功能在计算机的普及和应用中起到了重要的推动作用。今天，我们不但可以借助键盘、鼠标和显示器来操作计算机，还可以通过手写输入甚至语音输入与计算机进行对话。

在使用早期计算机时，用户遇到的另一个问题是，每次只能由一个用户独占整台计算机，并且只能让计算机做一件事情。如果有多个用户想使用计算机，或者一个用户想同时执行多个程序，则只能通过排队解决，即当一个程序运行结束之后，再开始执行另一个程序。

为了解决这一问题，人们为计算机设计了批处理功能，使得用户可以一次向计算机提交多个任务，让计算机自动地一个接一个地把整批任务执行完。20 世纪 50 年代中期，在美国通用汽车研究实验室诞生了世界上第一个操作系统。那是一个为 IBM 701 主机配备的批处理操作系统。当时人们似乎并没有考虑给它起一个名字。

批处理操作系统在一定程度上提高了计算机的使用效率，但并没有让用户实现同时执行多个程序的愿望。1962 年，美国麻省理工学院的考巴脱（F.J. Corbató）、达格特（M.M.Daggett）和戴利（R.C. Daley）发表了有关分时操作系统的研究论文。在美国麻省理工学院（MIT）计算中心诞生了世界上第一个兼容分时操作系统（CTSS）。

显而易见，有了分时操作系统的支持，多个用户能够同时使用一台计算机，因此计算机资源的利用率得到了进一步的提高。

至此，不妨从感性的角度谈一谈操作系统概念的基本意义。什么是操作系统呢？操作系统就是能够帮助用户方便地操作计算机，并能够提高计算机资源利用率的软件系统。作者认为，之所以称为“操作系统”，原因在于它是为方便用户高效地操作计算机而设计的。

20 世纪 60 年代末期，在 CTSS 的基础上，美国的麻省理工学院、贝尔实验室和通用电气公司联合开发出了称为多路信息和计算系统（Multics）的操作系统。在 Multics 的影响下， 贝尔实验室的汤普森（K.Thompson）和里奇（D. Ritchie）在 20 世纪 70 年代初期开发出了 UNIX 操作系统，并因此获得了 1983 年图灵奖。曾领导兼容分时系统和 Multics 开发的美国麻省理工学院的考巴脱教授在 1990 年也获得了图灵奖。

1973 年，基尔代尔（G.A. Kildall）为 Intel 8008 芯片开发了 8 位 CP/M 操作系统。这是基于英特尔芯片的计算机上的第一个操作系统。1980 年 4 月，美国西雅图计算机产

品公司的帕特森（T. Paterson）开发出 QDOS 操作系统，是 CP/M 的 16 位克隆版。1981 年 7 月，微软公司购买了 QDOS，随后以 PC DOS 和 MS DOS 的形象广泛应用于 IBM 的个人计算机中。在 MS DOS 的基础上，微软公司于 1985 年开发出了 Windows 1.0 版本，从此点燃了 Windows 操作系统的燎原之火。

在过去几十年，成百上千的操作系统涌现出来，UNIX 操作系统和 Windows 操作系统可以说是其中最具代表性的两大主要流派，在操作系统领域的贡献举足轻重。

微型化是计算机得以走进千家万户的重要因素之一。同时，在与微型化相对的方向上，高性能计算也是现代计算机技术追求的一个重要目标。在与分时处理相对的方向上，实时处理同样是一项重要的（某种意义上甚至是更重要的）应用需求。

操作系统在硬件技术和应用需求的驱动下不断向前发展。各类计算机都需要有与之相适应的操作系统为其提供支持。我们不但能看到支持桌面应用的 Windows XP 和 Linux 等操作系统，或者支持移动便携式应用的 Linux DA O/S、Palm OS 和 Windows CE 等操作系统，也能看到支持超级计算应用的 UNICOS 等操作系统；不但能看到支持分时应用的 UNIX 等操作系统，也能看到支持实时应用的 QNX 等操作系统。

计算机技术及其应用需求的多样性，造成了操作系统领域技术和产品的丰富多彩。而无论信息技术世界如何纷繁多变，为计算机系统提供基础支撑始终是操作系统永恒的主题。故此，纵然信息技术经历了几十年的沧海桑田，操作系统却一直是其华美乐章中多彩的主旋律。

2.3.2　操作系统的演化

操作系统已经经历了很长的一段发展历程，2.3.1 节已把操作系统的发展概况呈现给了大家，下面简要地归纳一下操作系统在发展过程中演化形成的主要类型。

1．批处理操作系统

批处理操作系统设计于 20 世纪 50 年代，目的是控制大型计算机。当时计算机十分庞大。它们用穿孔卡片输入数据，用行式打印机输出结果，磁带设备作为辅助存储介质。

每个执行的程序叫做作业。想要执行作业的程序员通过穿孔卡片将程序和数据输入计算机，并向操作室发出请求。程序员对系统没有控制和交互。穿孔卡片由操作员处理。如果程序运行成功，结果将传给程序员，如果不成功，则报错。

2．分时操作系统

为了有效使用计算机资源，引入了多道程序。它的做法是可以将多个作业装入存储器，并且仅当资源可用时分配给需要它的作业。例如，当一个程序正使用输入/输出设备时，CPU 则处于空闲状态，并可以供其他程序使用。

多道程序带来了分时的概念：资源可以被不同的作业共享。每个作业可以分到一段

时间来使用资源。因为计算机运行速度远远快于人的反应，所以分时系统对于用户是透明的，每个用户都感觉整个系统在为自己服务。

最终，利用分时技术的多道程序极大地提高了计算机系统的使用效率。但是，仍需要更加复杂的操作系统，它必须可以调度：给不同的程序分配资源并决定哪一个程序什么时候使用哪一种资源。

在这个时期，用户和计算机的关系也改变了。用户可以直接操作系统本身而不必通过操作员。一个新的术语也随之产生：进程。作业是要运行的程序，进程则是在存储器中等待资源的程序。

3．个人操作系统

当个人计算机产生后，需要有一类适合这类计算机的操作系统。于是，单用户操作系统，如 DOS（磁盘操作系统）产生了。

4．并行系统

对更快速度和更有效的要求导致了并行系统的设计：在同一计算机中安装了多个 CPU，每个 CPU 可以处理一个程序或程序的一部分。这意味着很多任务可以并行地处理而不再是串行处理。当然，这种操作系统要比单 CPU 的操作系统复杂得多。

5．分布式系统

网络化和交互式网络化的发展扩大了操作系统的内涵，产生了一种新的操作系统。以往必须在一台计算机上运行的作业现在可以由远隔千里的多台计算机共同完成。程序可以在一台计算机上运行一部分而在另一台计算机上运行另一部分，只要它们通过交互式网络如互联网互连。此外，资源可以是分布式的，程序需要的文件可能分布在世界的不同地方。分布式系统结合了以往系统的特点和新功能，如安全控制。

2.3.3　操作系统的构成

现在的操作系统十分复杂，它需要管理计算机系统内不同的资源。它像是一个有多个顶层经理的管理机构，每个经理负责自己部门的管理，并且相互协调合作。现代操作系统至少具有以下 4 种职能：存储管理、进程管理、设备管理、文件管理。就像很多组织有不直接在某个部门经理管理之下的部门一样，操作系统也有这样的部分，称为用户界面或命令解释程序，它负责操作系统与外界的联系（类似于公共关系部门）。

1．内存管理器

现在计算机操作系统的一个重要职责是存储管理。近年来，计算机的存储容量激增，同样，所处理的数据和程序也越来越大。存储分配必须进行管理以避免“内存不足”的错误。操作系统按照存储管理可以分为两大类：单道程序和多道程序。

1）单道程序

单道程序在过去很流行，但是它还是值得一提，因为它有助于理解多道程序。在单道程序里，大多数内存专用于单一的程序，仅有一小部分用来装载操作系统。在这种配置下，程序整体装入内存运行，运行结束后由其他程序取代。

2）多道程序

在多道程序中，同一时刻可以装入多个程序并且能够同时执行这些程序。CPU 轮流为它们服务。

从 20 世纪 60 年代开始，多道程序已经经过了一系列改进。有两种技术属于非交换类，这意味着程序在运行期间始终驻留在内存中。另外两种技术属于交换技术类。也就是说，在运行过程中，程序可以在内存和硬盘之间多次交换。

a. 分区调度

多道程序使用的第一种技术称为分区调度。在这种模式中，内存被分为不定长的几个分区。每个部分或分区保存一个程序。CPU 在各个程序之间交替服务。它由一个程序开始，执行一些指令直到有输入/输出操作或分配给那个程序的时限到达。CPU 保存最近执行指令的内存地址，然后转入下一个程序。对下一个程序采用同样的步骤执行。当为所有程序服务完毕后，CPU 再转回第一个程序。当然，CPU 可以进行优先级存取。

在这种技术下，每个程序完全载入内存，并占用了连续的地址。分区调度提高了 CPU 的使用效率，但仍有一些问题。

b. 分页调度

分页调度提高了分区调度的效率。在分页调度下，内存被分成大小相等的若干个部分，称之为帧；程序也被分为大小相等的部分，称之为页。页和帧的大小通常是一样的，并且与系统用于从存储设备中提取信息的块大小相等。

页被载入到内存的帧中。如果程序有 3 页，它就在内存中占用 3 个帧。在这种技术下，程序在内存中不必是连续的。两个连续的页可以占用内存里不连续的两个帧。分页调度对于分区调度的优势在于：一个需要 6 个帧的程序可以代替两个各占有不连续的 3 个帧的程序。新程序不必等到有 6 个连续的帧出现后再载入内存。

分页调度在一定程度上提高了效率，但整个程序仍需要在运行前全部载入内存。这意味着如果只有 4 个不连续帧，一个需要 6 个帧的程序是不能载入内存的。

c. 请求分页调度

分页调度不要求把程序装载在连续的内存中，但仍要求把程序整体载入内存中执行。请求分页调度去掉了后一种限制。在请求分页调度中，程序被分成页，但是页可以依次载入内存、运行，然后被另一个页代替。也就是说，内存可以同时载入多个程序的页。此外，来自同一个程序的连续页可以不必载入同一个帧，页可以被载入任何一个空闲帧。

3）虚拟内存

请求分页调度和请求分段调度意味着当程序执行时，一部分程序驻留在内存中，一部分则放在硬盘上。这就意味着，例如，10MB 内存可以运行 10 个程序。每个程序 3MB，一共 30MB，任一时候 10 个程序中的 10MB 在内存中，还有 20MB 在硬盘上。这里实际上只有 10MB 内存但却有 30MB 的虚拟内存。

2．进程管理器

操作系统的第二个功能是进程管理。

1）程序、作业和进程

现代操作系统关于指令集有三个术语：程序、作业和进程。尽管这些术语比较模糊，并且不同的操作系统对于它们的定义并不一致，我们还是可以非正式地定义这些术语。

- 程序。程序是由程序员编写的一组稳定的指令，存储在硬盘（或磁盘）上，它可能会也可能不会成为作业。
- 作业。从程序被选中执行到其运行结束并再次成为程序的这段过程中，程序称为作业。整个过程中，作业可能会或不会被执行。它或者驻留在硬盘上等待被装入内存，或者在内存中等待被 CPU 执行，或者驻留在硬盘或内存中等待输入/输出事件，或者在 CPU 执行时在内存中等待。在所有这些情况下的程序才称为作业。当作业执行完毕（正常或不正常），作业又变成程序并再次驻留在硬盘中，操作系统不再管理程序。需要注意的是：每个作业都是程序，但并不是所有的程序都是作业。
- 进程。进程是执行中的程序。该程序开始运行但还未结束。换句话说，进程是驻留在内存中的作业，它是从众多等待作业中选取出来并装入内存中的作业。进程可以处于运行状态或等待 CPU 调用。只要作业被装入内存就成为进程。需要注意的是：每个进程都是作业，而未必每个作业都是进程。

2）状态图

明白了程序怎样变成作业及作业怎样变成进程时，程序、作业、进程的关系也就很明显了。状态图显示了每个实体的不同状态。

当程序被操作系统选中时就成为作业并且成为保持状态。直至它被载入内存之前都保持这个状态。当内存可以整体或部分地载入这个程序时，作业转成就绪状态，并变成进程。它在内存中保持这个状态直至 CPU 执行它；这时它转成运行状态。当处于运行状态后，可能出现下面三种情况之一：① 进程运行直至它需要输入/输出；② 进程可能耗尽所分配的时间片；③ 进程终止。

在第一种情况下，进程进入等待状态并且等待直至输入/输出结束。在第二种情况下，直接进入就绪状态。在第三种情况下，它进入终止状态，并且不再是进程。进程进

入终止状态前在运行、等待、就绪状态中转换多次。注意：如果系统使用虚拟内存，并且在内存中将程序交换出或入，状态图可能更加复杂。

3）调度器

将作业或进程从一个状态转到另一个状态，进程管理器使用了两个调度器：作业调度器和进程调度器。

（1）作业调度器。作业调度器将作业从保持状态转入就绪状态，或者从运行状态转入终止状态。换句话说，作业调度器负责从作业中创建进程并终止进程。

（2）进程调度器。进程调度器将进程从一个状态转入另一个状态。当进程等待某事件发生时，它使这一进程从运行状态进入等待状态。当给进程分配的时间片用完时，这个进程从运行状态进入就绪状态。当触发事件发生时，进程将从等待状态进入就绪状态。当 CPU 准备运行这个进程时，进程调度器就让这个进程从就绪状态进入运行状态。

4）队列

事实上，会有很多的作业和进程相互竞争计算机资源。例如，当一些作业进入内存时，其他的就必须等待一直到有了可用空间。当一个进程正在使用 CPU，其他进程就必须等待直到 CPU 空闲为止。为处理多个进程和作业，进程管理器会使用队列（等待列表）。与每一个作业或进程相关联的是存有这些作业和进程信息的作业控制块或进程控制块。进程管理器在队列中存储的不是作业或进程，而是作业或进程控制块。作业和进程仍保存在内存或硬盘中，它们因为太大而无法被复制到队列里。作业控制块或进程控制块就是等待中的作业和进程的代表。

操作系统有很多队列。例如，作业队列、就绪队列和 I/O 队列。作业队列用来保存那些等待内存的作业；就绪队列用来保存那些已经在内存中准备好运行但在等待 CPU 的进程；I/O 队列用来保存那些正在等待 I/O 设备的进程（这里可以有多个 I/O 队列，每一个对应一个输入/输出设备）。

进程管理器可以用不同策略从队列中选择下一个作业或进程，可以是先入先出（First In First Out，FIFO）、最短长度优先、最高优先级等。

5）进程同步

所有进程管理的思想都是使得拥有不同资源的不同进程同步。只要资源可以被多个用户（进程）同时使用，那么它就有两种状态：死锁和饿死。下面简略说明一下这两种状态。

a. 死锁

先不给出死锁的正式定义，而是看一个例子。假定有两个进程 A 和 B，进程 A 占有了一个叫做 File1 的文件（File1 已经分配给了 A），而它只有得到另一个叫做 File2 的文件（A 已经请求了 File2）才能够释放 File1。进程 B 占有了一个叫做 File2 的文件（File2 已经分配给了 B），而它只有得到另一个叫做 File1 的文件（B 已经请求了

File1）才能够释放 File2。在大多数操作系统中，文件都是不可共享的。当文件被一个进程使用时，将不能再被其他的进程使用。在这种情况下，如果没有强制进程释放文件的防备措施，就会发生死锁。

死锁发生在这样的情况下：操作系统允许一个进程运行，而没有首先检查它所需要的资源是否准备好，以及是否允许这个进程占有资源直到它不需要为止。操作系统中需要有一些措施来防止死锁。一种解决方法是当所需资源不空闲时，不允许进程运行。但后面会发现这样做将导致另一个问题。另一种解决方法是限制进程占有资源的时间。

当操作系统没有对进程的资源进行限制时会发生死锁。

死锁不是经常发生，死锁发生需要 4 个必要条件：互斥（一个资源只能被一个进程占有）、资源占有（尽管并不使用资源，但进程占有着该资源直到有其他可用的资源）、抢先（操作系统不能临时对资源重新分配）、循环等待（所有的进程和资源包含在一个环里）。所有的 4 个条件都是死锁发生所必需的。但是它们只是必要条件（不是充分条件）。也就是说，对于死锁来说，它们必须出现，但它们并不一定能引起死锁。换句话说，如果它们其中之一没有出现，死锁就不会发生。这样就给我们提供了一种方法来防止或避免死锁：即不让它们中的某一条件发生。

b. 饿死

饿死是一种与死锁相反的情况。它发生在当操作系统对进程分配资源有太多限制时。例如，假如操作系统中规定一个进程只有在所需的所有资源都为其占有时才能执行。假设进程 A 需要两个文件 File1 和 File2。File1 正在被进程 B 使用，File2 正在被进程 E 使用。进程 B 首先终止而且释放了 Filel，但进程 A 一直不能执行因为 File2 一直没有被释放；与此同时，进程 C 由于只需要 File1 而被允许执行。这时进程 E 中止且释放 File2，但进程 A 还是不能执行，因为 File1 正被使用。

Dijkstra 介绍过一个经典的饿死问题。五个哲学家围坐在一个圆桌前，两个哲学家之间有一只筷子，每个哲学家需要用两只筷子来吃碗里的米饭。但一只或两只筷子可能被相邻的人使用。如果没有两只筷子同时可用的话，哲学家将被饿死。

3．设备管理器

设备管理器或输入/输出管理器负责访问输入/输出设备。在计算机系统中，输入/输出设备存在着数量和速度上的限制。由于这些设备与 CPU 和内存比起来速度要慢很多，所以当进程访问输入/输出设备时，在该段时间内这个设备对其他进程而言是不可用的。设备管理器负责有效使用输入/输出设备。

对设备管理器的详细讨论需要更深入地掌握有关操作系统的高级知识，这些都不在本书讨论范围内。但是可以在这里简要地列出设备管理器的职责。

（1）设备管理器不停地监视所有的输入/输出设备，以保证它们能够正常运行。管理器同样也需要知道什么时候设备已经完成一个进程的服务，而且准备为队列中的下一

个进程服务。

（2）设备管理器为每一个输入/输出设备维护一个队列，或者为类似的输入/输出设备维护一个或多个队列。例如，如果系统中有两个高速打印机，管理器能够分别用一个队列维护一个设备，或者用一个队列维护两个设备。

（3）管理器使用不同的方式来访问输入/输出设备。例如，可以用先入先出法来访问一个设备，而用最短长度优先来访问另一个设备。

4．文件管理器

现今的操作系统使用文件管理器来控制对文件的访问。下面简述一下文件管理器的职能。

（1）文件管理器控制对文件的访问。只有那些获得允许的才能够访问，访问方式也可以不同。例如，进程（用户也称为进程）也许可以读取文件，但却不允许写（改变）文件；另一个进程也许被允许执行文件，但却不允许查看文件的内容。

（2）文件管理器管理文件的创建、删除和修改。

（3）文件管理器可以给文件命名。

（4）文件管理器管理文件的存储：怎样存储、存在哪里等。

（5）文件管理器负责归档和备份。

5．用户界面

每个操作系统都有用户界面，即指用来接受用户（进程）的请求并向操作系统的其他部分解释这些请求的程序。一些操作系统的用户界面（如 UNIX）被称做命令解释程序（shell）。在其他操作系统中，则被称为窗口，以指明它是一个由菜单驱动的并有着 GUI（图形用户界面）的部件。

2.3.4　现代操作系统的特征

在过去数年中，操作系统的结构和功能逐步发展。但是近年来，许多新的设计要素引入到新操作系统及现有操作系统的新版本中，使操作系统产生了本质性的变化。这些现代操作系统的设计要素针对硬件中的新发展、新的应用程序和新的安全威胁。促使操作系统发展的硬件因素主要有：包含多处理器的计算机系统、高速增长的机器速度、高速网络连接和容量不断增加的各种存储设备。多媒体应用、互联网和 Web 访问、客户/服务器计算等应用领域也影响着操作系统的设计。在安全性方面，互联网的访问增加了潜在的威胁和更加复杂的攻击，如病毒、蠕虫和黑客技术，这些都对操作系统的设计产生了深远的影响。

对操作系统要求的变化速度不仅需要修改和增强现有的操作系统体系结构，而且需要有新的操作系统组织方法。在实验用和商用操作系统中有很多不同的方法和设计要素，大致可以分为以下几类：

- 微内核体系结构；
- 多线程；
- 对称多处理；
- 分布式操作系统；
- 面向对象设计。

至今，大多数操作系统都有一个单体内核（Monolithic Kernel），操作系统应该提供的大多数功能由该单体内核提供，常见功能包括系统调度、文件系统、网络通信、设备驱动、存储管理等。典型情况下，单体内核是作为一个进程实现的，所有元素都共享相同的地址空间。微内核体系结构（Microkernel Architecture）只给内核分配一些最基本的功能，包括地址空间、进程间通信（IPC，Inter-Process Communication）和基本的调度。其他的操作系统服务都是由运行在用户模式下且与其他应用程序类似的进程提供，这些进程可根据特定的应用和环境需求进行定制，有时也称这些进程为服务器。这种方法把内核和服务程序的开发分离开，可以为特定的应用程序或环境要求定制服务程序。微内核方法可以使系统结构的设计更加简单、灵活，很适合于分布式环境。实质上，微内核可以以相同的方式与本地和远程的服务进程交互，使分布式系统的构造更加方便。

多线程（Multithreading）技术是指把执行一个应用程序的进程划分成可以同时运行的多个线程。线程和进程有以下差别。

（1）线程：可分派的工作单元。它包括处理器上下文（包含程序计数器和栈指针）和栈中自己的数据区域（为允许子程序分支）。线程顺序执行，并且是可中断的，这样处理器就可以转到另一个线程。

（2）进程：一个或多个线程和相关系统资源（如包含数据和代码的存储器空间、打开的文件和设备）的集合。这紧密对应于一个正在执行的程序的概念。通过把一个应用程序分解成多个线程，程序员可以在很大程度上控制应用程序的模块性和应用程序相关事件的时间安排。

多线程对执行许多本质上独立、不需要串行处理的任务的应用程序是很有用的，如监听和处理很多客户请求的数据库服务器。在同一个进程中运行多个线程，在线程间来回切换所涉及的处理器开销要比在不同进程间进行切换所涉及的开销少。线程对构造进程是非常有用的。

到现在为止，大多数单用户的个人计算机和工作站基本上都只包含一个通用的微处理器。随着性能要求的不断增加及微处理器价格的不断降低，计算机厂商引进了拥有多个微处理器的计算机。为实现更高的有效性和可靠性，可使用对称多处理（SMP，Symmetric MultiProcessing）技术。对称多处理不仅指计算机硬件结构，而且指反映该硬件结构的操作系统行为。对称多处理计算机可以定义为具有以下特征的一个独立的计算机系统：① 有多个处理器；② 这些处理器共享同一个主存储器和 I/O 设备，它们之间通过通信总线或其他内部连接方案互相连接；③ 所有处理器都可以执行相同的功能

（因此称为对称）。

对称多处理操作系统可调度进程或线程到所有的处理器运行。对称多处理器结构比单处理器结构具有更多的潜在优势，包括如下几点。

（1）性能：如果计算机完成的工作可组织为让一部分工作并行完成，那么有多个处理器的系统将比只有一个同类型处理器的系统产生更好的性能。对于多道程序设计而言，一次只能执行一个进程，此时所有其他的进程都在等待处理器。对于多处理系统而言，多个进程可以分别在不同的处理器上同时运行。

（2）可用性：在对称多处理计算机中，由于所有的处理器都可以执行相同的功能，因而单个处理器的失败并不会使机器停止。相反，系统可以继续运行，只是性能有所降低。

（3）增量发展：用户可以通过添加额外的处理器来增强系统的功能。

（4）可扩展性：生产商可以根据系统配置的处理器的数量，提供一系列不同价格和性能特征的产品。

特别需要注意的是：这些只是潜在的优点，而不是确定的。操作系统必须提供发掘对称多处理计算机系统中并行性的工具和功能。

多线程和对称多处理总被放在一起讨论，但它们是两个独立的概念。即使在单处理器机器中，多线程对结构化的应用程序和内核进程也是很有用的。由于多个处理器可以并行运行多个进程，因而对称多处理计算机对非线程化的进程也是有用的。但是，这两个设施是互补的，一起使用将会更有效。

对称多处理技术一个极具吸引力的特征是多处理器的存在对用户是透明的。操作系统负责在多个处理器中调度线程或进程，并且负责处理器间的同步。另外一个不同的问题是给一群计算机（多机系统）提供单系统外部特征。在这种情况下，需要处理的是一群实体（计算机），每一个都有自己的主存、辅存和其他 I/O 模块。分布式操作系统使用户产生错觉，使多机系统好像具有一个单一的主存空间、辅存空间及其他的统一存取措施，如分布式文件系统。尽管集群正变得越来越流行，市场上也有很多集群产品，但分布式操作系统的技术发展水平落后于单处理器操作系统和对称多处理操作系统。

操作系统设计中的另一个改革是使用面向对象技术。面向对象设计的原理用于给小内核增加模块化的扩展。在操作系统一级，基于对象的结构使程序员可以定制操作系统，而不会破坏系统的完整性。面向对象技术还可使分布式工具和分布式操作系统的开发变得更加容易。

2.3.5　Linux和Windows操作系统结构

从操作系统的发展概况中，我们已经知道，UNIX 和 Windows 是两大具有广泛影响的操作系统流派，了解它们可增进对实际操作系统的认识，Linux 和 Windows 2000 是这

两大流派中的典型代表，下面简要勾画一下它们的体系结构。

1．Linux操作系统内核结构

图 2.14 显示了基于 IA-64 体系结构（如 Intel Itanium）的 Linux 操作系统内核的主要组件。图中显示了运行在内核之上的一些进程。每个方框表示一个进程，每条带箭头的曲线表示一个正在执行的线程。内核本身包括一组相互关联的组件，箭头表示主要的关联。底层的硬件也是一个组件集，箭头表示硬件组件被哪一个内核组件使用或控制。当然，所有的内核组件都在 CPU 上执行，但是为了简洁，这里没有显示它们的关系。

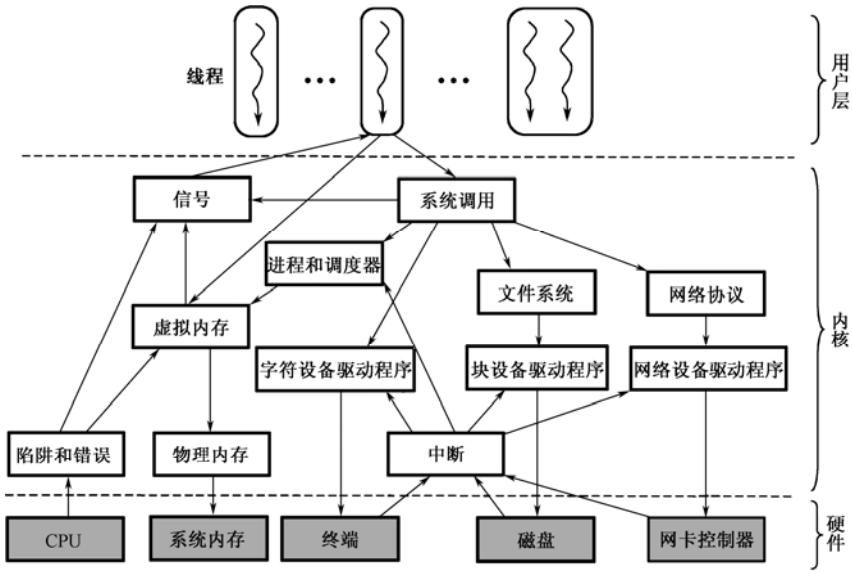


图 2.14　Linux 操作系统内核的主要组件

对主要的内核组件的简要介绍如下。

（1）信号（Signals）：内核通过信号调用进程。例如，信号用来通知进程某些错误，如被零除错误。

（2）系统调用（System Call）：进程是通过系统调用来请求系统服务的。一共有几百个系统调用，可以粗略地分为六类：文件系统、进程、调度、进程间通信、套接字（网络）和其他。

（3）进程和调度器（Processes and Scheduler）：创建、管理、调度进程。

（4）虚拟内存（Virtual Memory）：为进程分配和管理虚拟内存。

（5）文件系统（File Systems）：为文件、目录和其他与文件相关的对象提供一个全局的、层次化的命名空间，并提供文件系统函数。

（6）网络协议（Network Protocols）：为用户的 TCP/IP 协议套件提供套接字接口。

（7）字符设备驱动程序（Character Device Drivers）：管理向内核一次发送或接收一个字节数据的设备，如终端、调制解调器和打印机。

（8）块设备驱动程序（Block Device Drivers）：管理以块为单位向内核发送和接收数据的设备，如各种形式的辅存（磁盘、CD-ROM 等）。

（9）网络设备驱动程序（Network Device Drivers）：为网络接口卡和通信端口提供管理，它们负责连接到网桥或路由器之类的网络设备。

（10）陷阱和错误（Traps and Faults）：处理 CPU 产生的陷阱和错误，如内存错误。

（11）物理内存（Physical Memory）：管理实际内存中的内存页池并为虚拟内存分配内存页。

（12）中断（Interrupts）：处理来自外设的中断。

2．Windows 2000 操作系统体系结构

图 2.15 显示了 Windows 2000 的体系结构，该版本之后的各种 Windows 版本在本质上都有相同的结构。模块化结构给 Windows 2000 提供了相当大的灵活性，它被设计成可以在各种硬件平台上执行、可运行为各种其他操作系统编写的应用程序。

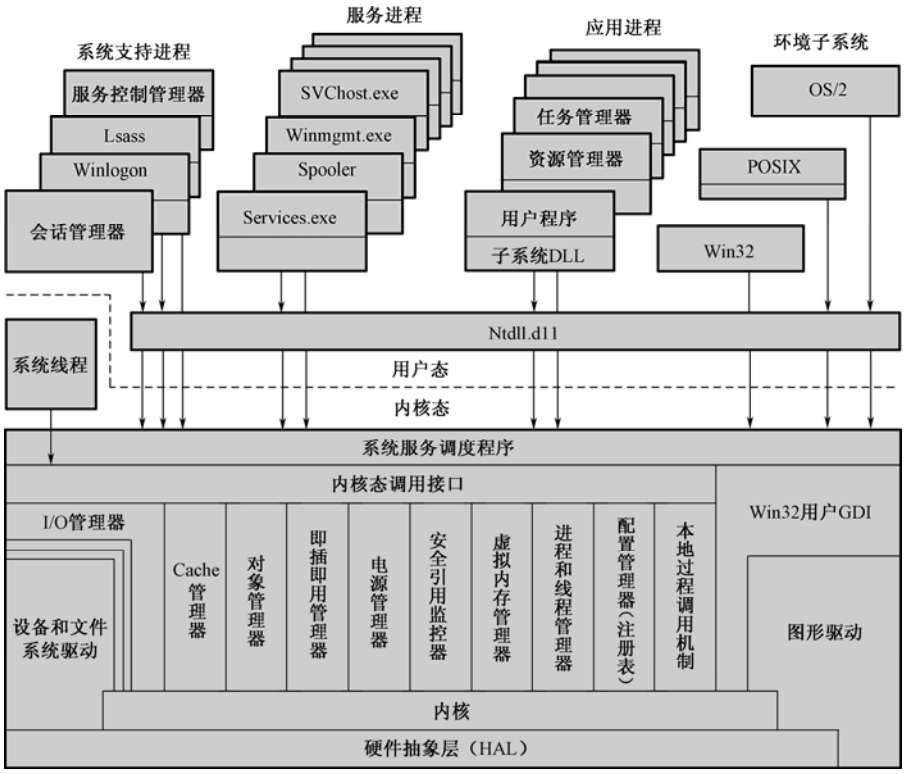


图 2.15　Windows 2000 体系结构

和其他操作系统一样，Windows 2000 把面向应用的软件和操作系统软件分开，后者包括在内核模式下运行的执行体、内核、设备驱动程序和硬件抽象层。内核模块软件可以访问系统数据和硬件，在用户模块下运行的其余软件则被限制访问系统数据。

Windows 并不是纯粹的微内核结构，微软将其称为改进的微内核结构。和纯粹的微内核结构一样，Windows 是高度模块化的。每个系统函数都正好由一个操作系统部件管理，操作系统的其余部分和所有应用程序通过相应的部件使用标准接口访问这个函数。关键的系统数据只能通过相应的函数访问。从理论上讲，任何模块都可以移动、升级或替换，而不需要重写整个系统或它的标准应用程序编程接口（API）。但是，与纯粹的微内核系统不同，Windows 的配置使许多微内核外的系统函数在内核模式下运行，其原因是性能要求。Windows 开发者发现，如果使用纯粹的微内核方法，则许多非微内核函数需要进行多次进程或线程切换、模式切换和使用附加的存储器缓冲区。

Windows 的内核模式组件包括以下类型。

（1）执行体（Executive）：包括操作系统基础服务，如内存管理、进程和线程管理、安全、I/O 及进程间通信。

（2）内核（Kernel）：包括操作系统中最常用和最基础的组件。内核管理包括线程调度、进程切换、异常和中断处理、多处理器同步。与执行体和用户级的其他部分不同，内核本身的代码并不在线程内执行。因此，内核是操作系统中唯一不可抢占或分页的一部分。

（3）硬件抽象层（Hardware Abstraction Layer，HAL）：在通用的硬件命令和响应与某一特定平台专用的命令和响应之间进行映射，它将操作系统从与平台相关的硬件差异中隔离出来，HAL 使得每个机器的系统总线、直接存储器访问（DMA）控制器、中断控制器、系统计时器和存储器模块对内核来说看上去都是相同的。它还对对称多处理（SMP）提供支持。

（4）设备驱动程序（Device Drivers）：包括文件系统和硬件设备驱动，它把用户 I/O 函数调用转换为特定硬件设备 I/O 请求。

（5）窗口和图形系统（Windowing and Graphics System）：实现图形用户界面函数（简称 GUI 函数），如处理窗口、用户界面控制和画图。

Windows 执行体包括一些特殊的系统函数模块，并为用户模式的软件提供 API。以下是对每个执行体模块的简单描述。

（1）I/O 管理器：提供了应用程序访问 I/O 设备的一个框架，还负责为进一步的处理分发合适的设备驱动程序。I/O 管理器实现了所有的 Windows I/O API，并实施了安全性、设备命名和文件系统（使用对象管理器）。

（2）Cache 管理器：通过使最近访问过的磁盘数据驻留在主存中以供快速访问，以及在更新后的数据发送到磁盘之前，通过在内存中保持一段很短的时间以延迟磁盘写操作，来提高基于文件的 I/O 性能。

（3）对象管理器：创建、管理和删除 Windows 执行体对象和用于表示诸如进程、线程和同步对象等资源的抽象数据类型。为对象的保持、命名和安全性设置实施统一的规则。对象管理器还创建对象句柄，对象句柄是由访问控制信息和指向对象的指针组

成的。

（4）即插即用管理器：决定并加载为支持一个特定的设备所需的驱动程序。

（5）电源管理器：调整各种设备间的电源管理，并且可以把处理器设置为休眠状态以达到节电的目的。

（6）安全引用监控器：实施访问确认和审计产生规则。Windows 面向对象模型允许统一且一致的安全视图，包含组成执行体的基本实体。因此，Windows 为所有受保护对象的确认访问和审核检查使用相同的例程，这些受保护对象包括文件、进程、地址空间和 I/O 设备。

（7）虚拟存管理器：把进程地址空间中的虚地址映射到计算机内存中的物理页。

（8）进程/线程管理器：创建和删除对象，跟踪进程和线程对象。

（9）配置管理器：负责执行和管理系统注册表，系统注册表是保存系统和每个用户参数设置的数据仓库。

（10）本地过程调用（LPC，Local Procedure Call）机制：在单个系统中，以类似于分布式处理中远程过程调用（RPC，Remote Procedure Call）的方式，在应用程序和执行子系统间强制实施客户/服务器关系。

Windows 支持如下四种基本的用户模式进程类型。

（1）特殊系统支持进程：包括未作为 Windows 操作系统一部分而提供的服务，如登录进程和会话管理程序。

（2）服务进程：诸如事件记录器之类的其他 Windows 服务。

（3）环境子系统：把本地的 Windows 服务提供给用户应用程序，因而提供一个操作系统环境或个性化支持。目前支持的子系统有 Win32、POSIX 和 OS/2。每个环境子系统包括动态链接库（DLL），它们把用户应用程序调用转换成 Windows 调用。

（4）用户应用程序：可以是以下五种类型之一：Win32、POSIX、OS/2、Windows 3.1 或 MS-DOS。

2.4　数据库系统基础

作为信息系统核心和基础的数据库技术得到越来越广泛的应用，从小型单项事务处理系统到大型信息系统，从联机事务处理到联机分析处理，从一般企业管理到计算机辅助设计与制造、计算机集成制造系统、电子政务、电子商务和地理信息系统等，越来越多新的应用领域采用数据库技术来存储和处理信息资源。

2.4.1　数据模型

数据模型（Data Model）是对现实世界数据特征的抽象，是用来描述数据、组织数据和对数据进行操作的。

由于计算机不可能直接处理现实世界中的具体事物，所以人们必须事先把具体事物转换成计算机能够处理的数据。也就是首先要数字化，把现实世界中具体的人、物、活动、概念用数据模型这个工具来抽象、表示和处理。

通俗地讲，数据模型就是现实世界的模拟。现有的数据库系统均是基于某种数据模型的。数据模型是数据库系统的核心和基础。了解数据模型的基本概念是学习数据库的基础。

1．两类数据模型

数据模型应满足三方面要求：一是能比较真实地模拟现实世界；二是容易被人所理解；三是便于在计算机上实现。一种数据模型要很好地、全面地满足这三方面的要求目前尚很困难。因此，在数据库系统中针对不同的使用对象和应用目的，采用不同的数据模型。

如同在建筑设计和施工的不同阶段需要不同的图纸一样，在开发实施数据库应用系统中，也需要使用不同的数据模型：概念模型、逻辑模型和物理模型。

根据模型应用的不同目的，可以将这些模型划分为两类，它们分别属于两个不同的层次。第一类是概念模型，第二类是逻辑模型和物理模型。

第一类是概念模型（Conceptual Model），也称信息模型，它按用户的观点来对数据和信息建模，主要用于数据库设计。

第二类中的逻辑模型主要包括层次模型（Hierarchical Model）、网状模型（Network Model）、关系模型（Relational Model）、面向对象模型（Object Oriented Model）和对象关系模型（Object Relational Model）等。它按计算机系统的观点对数据建模，主要用于数据库管理系统（DBMS）的实现。

第二类中的物理模型是对数据最低层的抽象，它描述数据在系统内部的表示方式和存取方法，在磁盘或磁带上的存储方式和存取方法，是面向计算机系统的。物理模型的具体实现是 DBMS 的任务，数据库设计人员要了解和选择物理模型，一般用户则不必考虑物理级的细节。

数据模型是数据库系统的核心和基础。各种机器上实现的 DBMS 软件都是基于某种数据模型或者说是支持某种数据模型的。

为了把现实世界中的具体事物抽象、组织为某一 DBMS 支持的数据模型，人们常常首先将现实世界抽象为信息世界，然后将信息世界转换为机器世界。也就是说，首先把现实世界中的客观对象抽象为某一种信息结构，这种信息结构并不依赖于具体的计算机系统，不是某一个 DBMS 支持的数据模型，而是概念级的模型；然后再把概念模型转换为计算机上某一 DBMS 支持的数据模型。

从现实世界到概念模型的转换是由数据库设计人员完成的，从概念模型到逻辑模型的转换可以由数据库设计人员完成，也可以用数据库设计工具协助设计人员完成，从逻

辑模型到物理模型的转换一般是由 DBMS 完成的。

2．数据模型的组成要素

一般地讲，数据模型是严格定义的一组概念的集合。这些概念精确地描述了系统的静态特性、动态特性和完整性约束条件（Integrity Constraints）。因此，数据模型通常由数据结构、数据操作和完整性约束三部分组成。

1）数据结构

数据结构描述数据库的组成对象及对象之间的联系。也就是说，数据结构描述的内容有两类：一类是与对象的类型、内容、性质有关的，如网状模型中的数据项、记录及关系模型中的域、属性、关系等；另一类是与数据之间联系有关的对象，如网状模型中的系型（Set Type）。

数据结构是刻画一个数据模型性质最重要的方面。因此，在数据库系统中，人们通常按照其数据结构的类型来命名数据模型。例如，层次结构、网状结构和关系结构的数据模型分别命名为层次模型、网状模型和关系模型。

总之，数据结构是所描述的对象类型的集合，是对系统静态特性的描述。

2）数据操作

数据操作是指对数据库中各种对象（型）的实例（值）允许执行的操作的集合，包括操作及有关的操作规则。

数据库主要有查询和更新（包括插入、删除、修改）两大类操作。数据模型必须定义这些操作的确切含义、操作符号、操作规则（如优先级）及实现操作的语言。

数据操作是对系统动态特性的描述。

3）完整性约束

数据的完整性约束条件是一组完整性规则。完整性规则是给定的数据模型中数据及其联系所具有的制约和依存规则，用以限定符合数据模型的数据库状态及状态的变化，以保证数据正确、有效、相容。

数据模型应该反映和规定本数据模型必须遵守的基本的通用的完整性约束条件。例如，在关系模型中，任何关系必须满足实体完整性和参照完整性两个条件（在关系数据库和数据库完整性等有关章节中将详细讨论这两个完整性约束条件）。

此外，数据模型还应该提供定义完整性约束条件的机制，以反映具体应用所涉及的数据必须遵守的特定的语义约束条件。例如，某大学的数据库中规定学生成绩如果有 6 门以上不及格将不能授予学士学位，教授的退休年龄是 65 周岁，男职工的退休年龄是 60 周岁，女职工的退休年龄是 55 周岁等。

2.4.2　概念模型

概念模型实际上是现实世界到机器世界的一个中间层次，用于信息世界的建模，是

现实世界到信息世界的第一层抽象，是数据库设计人员进行数据库设计的有力工具，也是数据库设计人员和用户之间进行交流的语言，因此概念模型一方面应该具有较强的语义表达能力，能够方便、直接地表达应用中的各种语义知识，另一方面它还应该简单、清晰、易于被用户理解。

1．信息世界中的基本概念

信息世界涉及的概念主要有如下几个。

（1）实体（Entity）。客观存在并可相互区别的事物称为实体。实体可以是具体的人、事、物，也可以是抽象的概念或联系，如一个职工、一个学生、一个部门、一门课、学生的一次选课、一次订货、老师与院系的工作关系（即某位老师在某院系工作）等都是实体。

（2）属性（Attribute）。实体所具有的某一特性称为属性。一个实体可以由若干个属性来刻画。例如，学生实体可以由学号、姓名、性别、出生年月、所在院系、入学时间等属性组成。（94002268，张山，男，197605，计算机系，1994）这些属性组合起来表征了一个学生。

（3）码（Key）。唯一标识实体的属性集称为码，如学号是学生实体的码。

（4）域（Domain）。域是一组具有相同数据类型的值的集合。属性的取值范围来自某个域。例如，学号的域为 8 位整数，姓名的域为字符串集合，学生年龄的域为整数，性别的域为（男，女）。

（5）实体型（Entity Type）。具有相同属性的实体必然具有共同的特征和性质。用实体名及其属性名的集合来抽象和刻画同类实体，称为实体型。例如，学生（学号，姓名，性别，出生年月，所在院系，入学时间）就是一个实体型。

（6）实体集（Entity Set）。同一类型实体的集合称为实体集。例如，全体学生就是一个实体集。

（7）联系（Relationship）。在现实世界中，事物内部及事物之间是有联系的，这些联系在信息世界中反映为实体（型）内部的联系和实体（型）之间的联系。实体内部的联系通常指组成实体的各属性之间的联系；实体之间的联系通常指不同实体集之间的联系。

2．两个实体型之间的联系

两个实体型之间的联系可以分为以下三种。

1）一对一联系（1:1）

如果对于实体集 *A* 中的每一个实体，实体集 *B* 中至多有一个（也可以没有）实体与之联系，反之亦然，则称实体集 *A* 与实体集 *B* 具有一对一的联系，记为 1:1。

例如，学校里面，一个班级只有一个正班长，而一个班长只在一个班中任职，则班级与班长之间具有一对一联系。

2）一对多联系（1:n）

如果对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体（$n \geqslant 0$）与之联系，反之，对于实体集 B 中的每一个实体，实体集 A 中至多只有一个实体与之联系，则称实体集 A 与实体集 B 有一对多联系，记为 1:n。

例如，一个班级中有若干名学生，而每个学生只在一个班级中学习，则班级与学生之间具有一对多联系。

3）多对多联系（m:n）

如果对于实体集 A 中的每一个实体，实体集 B 中有 n 个实体（$n \geqslant 0$）与之联系，反之，对于实体集 B 中的每一个实体，实体集 A 中也有 m 个实体（$m \geqslant 0$）与之联系，则称实体集 A 与实体集 B 具有多对多联系，记为 m:n。

例如，一门课程同时有若干名学生选修，而一名学生可以同时选修多门课程，则课程与学生之间具有多对多联系。

实际上，一对一联系是一对多联系的特例，而一对多联系又是多对多联系的特例。

3．概念模型的表示方法

概念模型是对信息世界建模，所以概念模型应该能够方便、准确地表示出信息世界中的常用概念。概念模型的表示方法很多，其中最为著名、最为常用的是陈（P.P.S. Chen）于 1976 年提出的实体−联系方法（Entity-Relationship Approach）。该方法用 E-R 图（E-R Diagram）来描述现实世界的概念模型，E-R 方法也称为 E-R 模型。

E-R 图提供了表示实体型、属性和联系的方法。

- 实体型：用矩形表示，矩形框内写明实体名。
- 属性：用椭圆形表示，并用无向边将其与相应的实体型连接起来。
- 联系：用菱形表示。菱形框内写明联系名，并用无向边分别与有关实体型连接起来，同时在无向边旁标上联系的类型（1:1、1:n 或 m:n）。

须注意的是：如果一个联系具有属性，则这些属性也要用无向边与该联系连接起来。

2.4.3　关系模型

关系模型是目前最重要的一种数据模型。关系数据库系统采用关系模型作为数据的组织方式。

1970 年，美国 IBM 公司 San Jose 研究室的研究员柯德（E.F. Codd）首次提出了数据库系统的关系模型，开创了数据库关系方法和关系数据理论的研究，为数据库技术奠定了理论基础。由于柯德的杰出工作，他于 1981 年获得 ACM 图灵奖。

20 世纪 80 年代以来，计算机厂商推出的数据库管理系统几乎都支持关系模型，非关系系统的产品大都加上了关系接口。数据库领域当前的研究工作也都是以关系方法为基础的。

1．关系模型的数据结构

关系模型与以往的模型不同，它是建立在严格的数学概念的基础上的。这里只简单勾画一下关系模型。从用户的角度看，关系模型由一组关系组成。每个关系的数据结构是一张规范化的二维表。现在以学生登记表（见图 2.16）为例，介绍关系模型中的一些术语。

学生登记表

学号	姓名	年龄	性别	系别	年级
2007621	陈晓娟	18	女	计算机	2007
2007702	张海涛	19	男	物理	2007
2007805	郭文慧	17	女	数学	2007
…	…	…	…	…	…

图 2.16　关系模型的数据结构

（1）关系（Relation）：一个关系对应一张二维表，如图 2.16 所示的这张学生登记表。

（2）元组（Tuple）：表中的一行即为一个元组。

（3）属性（Attribute）：表中的一列即为一个属性，给每一个属性起一个名称即属性名。例如，学生登记表有 6 列，对应 6 个属性（学号，姓名，年龄，性别，系名和年级）。

（4）码（Key）：也称为码键。表中的某个属性组，它可以唯一确定一个元组，如图 2.16 所示的学号，可以唯一确定—个学生，也就成为本关系的码。

（5）域（Domain）：属性的取值范围，如人的年龄一般在 1～150 岁之间，大学生年龄属性的域是（14～38），性别的域是（男，女），系名的域是一个学校所有系名的集合。

（6）分量：元组中的一个属性值。

（7）关系模式：对关系的描述，一般表示为

关系名（属性 1，属性 2，…，属性 *n*）

例如，上面的关系可描述为

学生（学号，姓名，年龄，性别，系名，年级）

在关系模型中，实体及实体间的联系都是用关系来表示的。例如，学生、课程、学生与课程之间的多对多联系在关系模型中可以表示如下：

学生（学号，姓名，年龄，性别，系名，年级）

课程（课程号，课程名，学分）

选修（学号，课程号，成绩）

关系模型要求关系必须是规范化的，即要求关系必须满足一定的规范条件，这些规范条件中最基本的一条就是：关系的每个分量必须是一个不可分的数据项，也就是说，

不允许表中还有表。

2．关系模型的操作与完整性约束

关系模型的操作主要包括查询、插入、删除和更新数据。这些操作必须满足关系的完整性约束条件。关系的完整性约束条件包括三大类：实体完整性、参照完整性和用户定义的完整性。

关系模型中的数据操作是集合操作，操作对象和操作结果都是关系，即若干元组的集合，而不像格式化模型中那样是单记录的操作方式。另一方面，关系模型把存取路径向用户隐蔽起来，用户只要指出“干什么”或“找什么”，不必详细说明“怎么干”或“怎么找”，从而大大地提高了数据的独立性，提高了效率。

在关系数据库中，典型的操作包括：插入、删除、更新、选择、投影、连接、并、交和差等。

插入操作是一种一元操作，它应用于一个关系，其操作是在表中插入新的元组。

删除操作也是一元操作，它根据要求删去表中相应的元组。

更新操作也是一元操作，它应用于一个关系，用来更新元组中的部分属性值。

选择操作也是一元操作，它应用于一个关系并产生另外一个新关系。新关系中的元组（行）是原关系中元组的子集。选择操作根据要求从原关系中选择部分元组。该操作中属性（列）的数量保持不变。

投影操作也是一元操作，它用于一个关系并产生另外一个关系。新关系中的属性（列）是原关系中属性的子集。投影操作所得到的新关系中的元组属性减少。在这个操作中元组（行）的数量保持不变。

连接操作是二元操作。它基于共有的属性把两个关系组合起来。连接操作十分复杂并有很多变化。

并操作也是二元操作。它将两个关系组合成一个新的关系。不过这里对这两个关系有一个限制，即它们必须有相同的属性。根据集合论中的定义，并操作创建的新关系中的每一个元组在第一个关系、第二个关系，或者在两关系中皆有。

交操作也是二元操作，它对两关系进行操作，创建一个新关系。和并操作一样，进行交操作的两个关系必须有相同的属性。根据集合论中的定义，交操作创建的新关系中的每一个元组必须是两个原关系中共有的成员。

差操作也是二元操作，它应用于具有相同属性的两个关系。生成的关系中的元组是那些存在于第一个关系中而不在第二个关系中的元组。

3．关系模型的存储结构

在关系数据模型中，实体及实体间的联系都用表来表示。在关系数据库的物理组织中，有的 DBMS 中一个表对应一个操作系统文件，有的 DBMS 从操作系统获得若干个大的文件，自己设计表索引等存储结构。

4．结构化查询语言

结构化查询语言（SQL）是美国国家标准协会（ANSI）和国际标准组织（ISO）用于关系数据库的标准化语言。这是一种描述性（不是过程化）的语言，这意味着使用者不需要编写一步步的详细程序而只须声明它。结构化查询语言最早于 1979 年首次被 Oracle 公司实现，之后有了更多的新版本 SQL。

下面是与关系模型的操作有关的结构化查询语言中的常用语句。

插入操作使用如下格式。values 子句定义了与插入的元组相对应的属性值。

```
INSERT INTO RELATION-NAME
VALUES {…, …, …}
```

删除操作的一般格式如下。删除需要满足的条件定义在 WHERE 子句中。

```
DELETE FROM RELATION-NAME
WHERE criteria
```

更新操作使用如下的格式。需要更新的属性在 set 子句中定义。where 子句中则定义了要更新的元组须满足的条件。

```
UPDATE     RELATION-NAME
SET attribute1 = value1, attribute2 ＝ value2, …
WHERE     criteria
```

选择操作使用如下的格式。*表示选择所有的属性。

```
SELECT *
FROM    RELATION-NAME
WHERE     criteria
```

投影操作使用如下的格式，新关系的列名被显式地列出。

```
SELECT    attribute-list
FROM    RELATION-NAME
```

连接操作使用如下的格式，属性列表是原来两个输入关系的属性的组合。where 子句明确定义了用于连接的共有属性。

```
SELECT    attribute-list
FROM    RELATION1, RELATION2
WHERE     criteria
```

并操作使用如下的辂式。同样，这里*表示选择所有的属性。

```
SELECT    *
FROM    RELATION1
UNION
SELECT    *
FROM RELATION2
```

交操作使用如下的格式。同样，这里*表示选择所有的属性。

```
SELECT　*
FROM　RELATION1
intersection
SELECT　*
FROM　RELATION2
```

差操作使用如下的格式。同样，这里*表示选择所有的属性。

```
SELECT　*
FROM　RELATIONl
minus
SELECT　*
FROM　RELATION2
```

SQL 还允许用户组合使用上述语句以便从数据库中获得更复杂的信息。

2.4.4　数据库系统结构

认识数据库系统的结构可以有多种不同的层次或不同的角度。

从数据库管理系统角度看，数据库系统通常采用三级模式结构，这是数据库管理系统内部的系统结构。

从数据库最终用户角度看，数据库系统的结构分为单用户结构、主从式结构、分布式结构、客户机/服务器、浏览器/应用服务器/数据库服务器多层结构等，这是数据库系统外部的体系结构。

1．数据库系统模式的概念

在数据模型中有“型”（Type）和“值”（Value）的概念。型是指对某一类数据的结构和属性的说明，值是型的一个具体赋值。例如，学生记录定义为（学号，姓名，性别，系别，年龄，籍贯）这样的记录型，而（900201，李明，男，计算机，22，江苏）则是该记录型的一个记录值。

模式（Schema）是数据库中全体数据的逻辑结构和特征的描述，它仅涉及型的描述，不涉及具体的值。模式的一个具体值称为模式的一个实例（Instance）。同一个模式可以有很多实例。

例如，在学生选课数据库模式中，包含学生记录、课程记录和学生选课记录，则 2003 年有一个学生数据库的实例，该实例包含了 2003 年学校中所有学生的记录（如果某校有 10 000 名学生，则有 10 000 个学生记录）、学校开设的所有课程的记录和所有学生选课的记录。

2002 年度学生数据库模式对应的实例与 2003 年度学生数据库模式对应的实例是不同的。实际上，2003 年度学生数据库的实例也会随时间变化，因为在此年度有的学生可能退学，有的学生可能转系。各个时刻学生选课数据库的实例是不同的、在变化的，而

不变的是学生选课数据库模式。

模式是相对稳定的，而实例是相对变动的，因为数据库中的数据是在不断更新的。模式反映的是数据的结构及其联系，而实例反映的是数据库在某一时刻的状态。

虽然实际的数据库管理系统产品种类很多，它们支持不同的数据模型，使用不同的数据库语言，建立在不同的操作系统之上，数据的存储结构也各不相同，但它们在体系结构上通常都具有相同的特征，即采用三级模式结构（早期微型计算机上的小型数据库系统除外）并提供两级映像功能。

2．数据库系统的三级模式结构

数据库系统的三级模式结构是指数据库系统是由外模式、模式和内模式三级构成的，如图 2.17 所示。

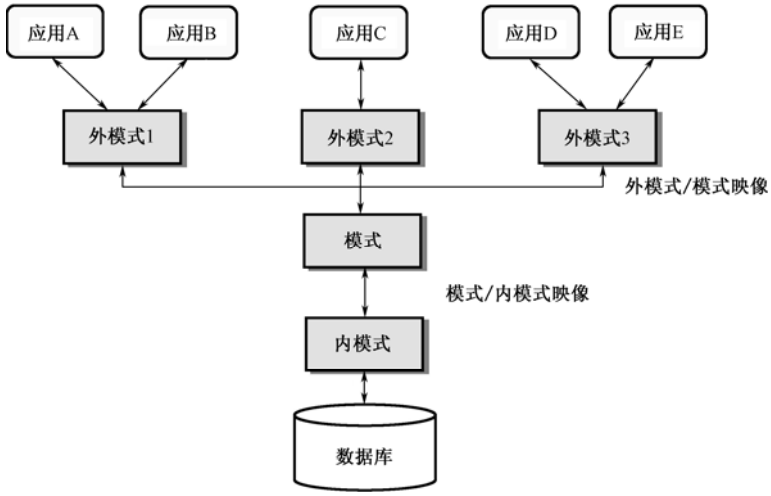


图 2.17　数据库系统的三级模式结构

1）模式（Schema）

模式也称逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。它是数据库系统模式结构的中间层，既不涉及数据的物理存储细节和硬件环境，也与具体的应用程序、所使用的应用开发工具及高级程序设计语言（如 C、COBOL、Fortran）无关。

模式实际上是数据库数据在逻辑级上的视图。一个数据库只有一个模式。数据库模式以某一种数据模型为基础，统一综合地考虑了所有用户的需求，并将这些需求有机地结合成一个逻辑整体。定义模式时不仅要定义数据的逻辑结构，例如，数据记录由哪些数据项构成，数据项的名字、类型、取值范围等，而且要定义数据之间的联系，定义与数据有关的安全性、完整性要求。

DBMS 提供模式描述语言（模式 DDL）来严格地定义模式。

2）外模式（External Schema）

外模式也称子模式（Subschema）或用户模式，它是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。

外模式通常是模式的子集。一个数据库可以有多个外模式。由于它是各个用户的数据视图，如果不同的用户在应用需求、看待数据的方式、对数据保密的要求等方面存在差异，则其外模式描述就是不同的。即使对于模式中同一数据，在外模式中的结构、类型、长度、保密级别等都可以不同。另一方面，同一外模式也可以为某一用户的多个应用程序所使用，但一个应用程序只能使用一个外模式。

外模式是保证数据库安全性的一个有力措施。每个用户只能看见和访问所对应的外模式中的数据，数据库中的其余数据是不可见的。

DBMS 提供子模式描述语言（子模式 DDL）来严格地定义子模式。

3）内模式（Internal Schema）

内模式也称存储模式（Storage Schema），一个数据库只有一个内模式。它是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式。例如，记录的存储方式是堆存储，还是按照某个（些）属性值的升（降）序存储，还是按照属性值聚簇（Cluster）存储；索引按照什么方式组织，是 B+树索引，还是 hash 索引；数据是否压缩存储，是否加密；数据的存储记录结构有何规定，如定长结构或变长结构，一个记录不能跨物理页存储；等等。

DBMS 提供内模式描述语言（内模式 DDL 或存储模式 DDL）来严格地定义内模式。

3．数据库的二级映像功能与数据独立性

数据库系统的三级模式是对数据的 3 个抽象级别，它把数据的具体组织留给 DBMS 管理，使用户能逻辑地、抽象地处理数据，而不必关心数据在计算机中的具体表示方式与存储方式。为了能够在系统内部实现这 3 个抽象层次间的联系和转换，数据库管理系统在这三级模式之间提供了两层映像：

- 外模式/模式映像；
- 模式/内模式映像。

正是这两层映像保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。

1）外模式/模式映像

模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。对应于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式/模式映像，它定义了该外模式与模式之间的对应关系。这些映像定义通常包含在各自外模式的描述中。

当模式改变时（如增加新的关系、新的属性、改变属性的数据类型等），由数据库管理员对各个外模式/模式的映像做相应的改变，可以使外模式保持不变。应用程序是依据数据的外模式编写的，进而应用程序不必修改，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

2）模式/内模式映像

数据库中只有一个模式，也只有一个内模式，所以模式/内模式映像是唯一的，它定义了数据全局逻辑结构与存储结构之间的对应关系。例如，说明逻辑记录和字段在内部是如何表示的。该映像定义通常包含在模式描述中。当数据库的存储结构改变了（如选用了另一种存储结构），由数据库管理员对模式/内模式映像做相应的改变，可以使模式保持不变，进而应用程序也不必改变。保证了数据与程序的物理独立性，简称数据的物理独立性。

在数据库的三级模式结构中，数据库模式即全局逻辑结构是数据库的中心与关键，它独立于数据库的其他层次。因此设计数据库模式结构时应首先确定数据库的逻辑模式。

数据库的内模式依赖于它的全局逻辑结构，但独立于数据库的用户视图，即外模式，也独立于具体的存储设备。它将全局逻辑结构中定义的数据结构及其联系按照一定的物理存储策略进行组织，以达到较好的时间与空间利用效率。

数据库的外模式面向具体的应用程序，它定义在逻辑模式之上，但独立于存储模式和存储设备。当应用需求发生较大变化使相应外模式不能满足其视图要求时，该外模式就要做相应的改动，所以设计外模式时应充分考虑到应用的扩充性。

特定的应用程序是在外模式描述的数据结构上编制的，它依赖于特定的外模式，与数据库的模式和存储结构独立。不同的应用程序有时可以共用同一个外模式。数据库的二级映像保证了数据库外模式的稳定性，进而从底层保证了应用程序的稳定性，除非应用需求本身发生变化，否则应用程序一般都不需要修改。

数据与程序之间的独立性使得数据的定义和描述可以从应用程序中分离出去。另外，由于数据的存取由 DBMS 管理，用户不必考虑存取路径等细节，从而简化了应用程序的编制，大大减少了应用程序的维护和修改。

2.4.5　数据库系统的组成

数据库系统一般由数据库、数据库管理系统（及其开发工具）、应用系统和数据库管理员组成。下面分别介绍这几个部分内容。

1．硬件平台及数据库

由于数据库系统数据量很大，加之 DBMS 丰富的功能，使得自身的规模也很大。因此，整个数据库系统对硬件资源提出了较高的要求，这些要求如下：

（1）要有足够大的内存来存放操作系统、DBMS 的核心模块、数据缓冲区和应用

程序；

（2）要有足够的大的磁盘或磁盘阵列等设备存放数据库，有足够多的磁带（或光盘）进行数据备份；

（3）要求系统有较高的通道能力，以提高数据传送率。

2．软件

数据库系统的软件主要包括：

（1）DBMS，DBMS 是为数据库的建立、使用和维护配置的系统软件；

（2）支持 DBMS 运行的操作系统；

（3）具有与数据库接口的高级语言及其编译系统，便于开发应用程序；

（4）以 DBMS 为核心的应用开发工具，应用开发工具是系统为应用开发人员和最终用户提供的高效率、多功能的应用生成器、第四代语言等各种软件工具，它们为数据库系统的开发和应用提供了良好的环境；

（5）为特定应用环境开发的数据库应用系统。

3．人员

开发、管理和使用数据库系统的人员主要有：数据库管理员、系统分析员和数据库设计人员、应用程序员和最终用户。不同的人员涉及不同的数据抽象级别，具有不同的数据视图，各种人员的数据库视图如图 2.18 所示，其各自的职责分别如下所述。

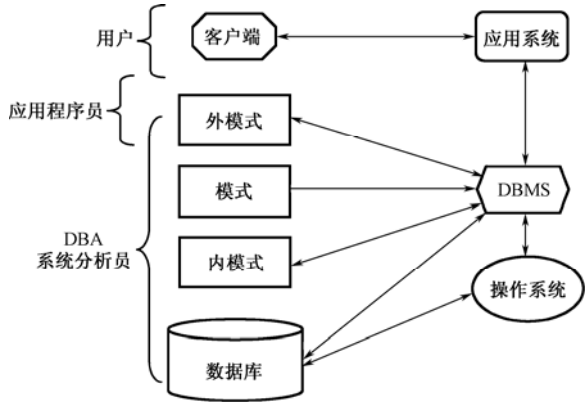


图 2.18　各种人员的数据库视图

1）数据库管理员（DBA，DataBase Administrator）

在数据库系统环境下，有两类共享资源：一类是数据库；另一类是数据库管理系统软件。因此，需要有专门的管理机构来监督和管理数据库系统。DBA 则是这个机构中的一个（组）人员，负责全面管理和控制数据库系统。具体职责包括：

（1）决定数据库中的信息内容和结构。数据库中要存放哪些信息，DBA 要参与决策。因此，DBA 必须参加数据库设计的全过程，并与用户、应用程序员、系统分析员密

切合作、共同协商，搞好数据库设计。

（2）决定数据库的存储结构和存取策略。DBA 要综合各用户的应用要求，和数据库设计人员共同决定数据的存储结构和存取策略，以求获得较高的存取效率和存储空间利用率。

（3）定义数据的安全性要求和完整性约束条件。DBA 的重要职责是保证数据库的安全性和完整性。因此，DBA 负责确定各个用户对数据库的存取权限、数据的保密级别和完整性约束条件。

（4）监控数据库的使用和运行。DBA 还有一个重要职责就是监视数据库系统的运行情况，及时处理运行过程中出现的问题。例如，系统发生各种故障时，数据库会因此遭到不同程度的破坏，DBA 必须在最短的时间内将数据库恢复到正确状态，并尽可能不影响或少影响计算机系统其他部分的正常运行。为此，DBA 要定义和实施适当的后备和恢复策略，如周期性地转存数据、维护日志文件等。

（5）数据库的改进和重组重构。DBA 还负责在系统运行期间监视系统的空间利用率、处理效率等性能指标，对运行情况进行记录、统计分析，依靠工作实践并根据实际应用环境不断改进数据库设计。不少数据库产品都提供了对数据库运行状况进行监视和分析的工具，DBA 可以使用这些软件完成这项工作。

另外，在数据运行过程中，大量数据不断插入、删除、修改，时间一长，会影响系统的性能。因此，DBA 要定期对数据库进行重组织，以提高系统的性能。

当用户的需求增加和改变时，DBA 还要对数据库进行较大的改造，包括修改部分设计，即数据库的重构造。

2）系统分析员和数据库设计人员

系统分析员负责应用系统的需求分析和规范说明，要和用户及 DBA 合作，确定系统的硬件和软件配置，并参与数据库系统的概要设计。

数据库设计人员负责数据库中数据的确定、数据库各级模式的设计。数据库设计人员必须参加用户需求调查和系统分析，然后进行数据库设计。在很多情况下，数据库设计人员就由数据库管理员担任。

3）应用程序员

应用程序员负责设计和编写应用系统的程序模块，并进行调试和安装。

4）用户

这里的用户是指最终用户（End User）。最终用户通过应用系统的用户接口使用数据库。常用的接口方式有浏览器、菜单驱动、表格操作、图形显示、报表书写等。

最终用户可以分为以下三类。

第一类：偶然用户。这类用户不经常访问数据库，但每次访问数据库时往往需要不同的数据库信息，这类用户一般是企业或组织机构的中高级管理人员。

第二类：简单用户。数据库的多数最终用户都是简单用户。其主要工作是查询和更

新数据库，一般都是通过应用程序员精心设计的具有友好界面的应用程序存取数据库。银行的职员、航空公司的机票预定工作人员、旅馆总台服务员等都属于这类用户。

第三类：复杂用户。复杂用户包括工程师、科学家、经济学家、科学技术工作者等具有较高科学技术背景的人员。这类用户一般都比较熟悉数据库管理系统的各种功能，能够直接使用数据库语言访问数据库，甚至能够基于数据库管理系统的 API 编制自己的应用程序。

2.5　本章小结

计算机系统是系统安全的载体，为了更好地从整个计算机系统的角度讨论安全问题，本章简要回顾了计算机硬件、操作系统和数据库系统等方面的基础知识。首先，以一个程序员的身份领略了计算机系统的基本面貌，然后，通过传统的方式品味了计算机组成、操作系统和数据库系统等方面的精髓。

尽管 hello 程序是一个非常简单的程序，但是为了完成它的执行，计算机系统的各个主要组成部分都需要协调工作。围绕 hello 程序的执行过程，我们认识了计算机硬件中的总线、I/O 设备、主存和处理器等主要部件的功能和工作原理，了解了高速缓冲存储器的作用及存储设备层次结构的基本情况，了解了操作系统支持应用程序运行的基本方法及操作系统中的进程、线程、虚拟存储器和文件等基本概念，还了解了网络设备的基本情况和程序在网络上运行的基本方法。

计算机系统的硬件组成包括中央处理单元、主存储器和输入/输出子系统，各个子系统通过总线进行连接，通过地址进行定位。

中央处理单元用于数据的运算，它包含算术逻辑单元、控制单元和寄存器三个组成部分。算术逻辑单元用于进行算术运算和逻辑运算，寄存器是用来临时存放数据的高速独立的存储单元，控制单元类似于人脑中控制身体各部分运动的区域，它通过线路的开（高电平）或关（低电平）来实现控制。

存储器有 RAM 和 ROM 两种类型。RAM 是计算机中主存的主要组成部分，用户可读/写 RAM，储存在 RAM 中的信息在断电后将被删除。ROM 的内容是由制造商写进去的，用户只能读但不能写，储存在 ROM 中的数据在切断电源后不会丢失。

输入/输出子系统实现计算机与外界的通信，并在断电的情况下存储程序和数据，输入/输出设备可以分为非存储设备和存储设备两大类。非存储设备使 CPU/内存可以与外界通信，但它不能存储信息。存储设备可以存储大量的信息以备后用，通常可分为磁介质存储设备和光介质存储设备两种类型。

操作系统是计算机硬件和用户的接口，它使其他程序更加方便有效地执行，并能方便地对计算机硬件和软件资源进行访问。操作系统的两个主要设计目标是有效地使用硬件并容易地使用资源。

在发展过程中，操作系统演化形成了批处理操作系统、分时操作系统、单用户操作系统、并行操作系统和分布式操作系统等多种主要类型。

现代操作系统至少具有存储管理、进程管理、设备管理和文件管理等四种职能，其构成通常包含内存管理器、进程管理器、设备管理器、文件管理器和用户界面等主要部分。

现代操作系统的特征主要体现在微内核体系结构、多线程、对称多处理、分布式处理和面向对象设计等方面。

传统的大多数操作系统拥有单体内核结构，微内核体系结构只给内核分配一些最基本的功能，如地址空间、进程间通信和基本的调度等，其他的操作系统服务都由运行在用户模式下的进程提供。

多线程技术把执行一个应用程序的进程划分成可以同时运行的多个线程。对称多处理计算机拥有多个共享同一个主存储器和 I/O 设备的处理器，对称多处理操作系统可调度进程或线程到所有的处理器运行，多处理器的存在对用户是透明的。

现有数据库系统均基于某种数据模型，数据模型是数据库系统的核心和基础。数据模型是对现实世界数据特征的抽象，用来描述数据、组织数据和对数据进行操作。数据模型可以划分为两类：第一类是概念模型，第二类是逻辑模型和物理模型。

概念模型按用户的观点来对数据和信息建模，主要用于数据库设计。概念模型最常用的表示方法是实体−联系方法，即 E-R 图方法。逻辑模型主要包括层次模型、网状模型、关系模型、面向对象模型和对象关系模型等，它按计算机系统的观点对数据建模，主要用于数据库管理系统的实现。物理模型是对数据最低层的抽象，它描述数据在系统内部的表示方式和存取方法，是面向计算机系统的。

关系模型是目前最重要的一种数据模型（逻辑模型），关系数据库系统采用关系模型作为数据的组织方式。关系模型由一组关系组成，每个关系的数据结构是一张规范化的二维表。关系数据模型主要包括查询、插入、删除和更新数据等类型的操作。结构化查询语言（SQL）是在关系数据库中使用的标准化语言。

在数据模型中，有“型”和“值”的概念。模式是数据库中全体数据的逻辑结构和特征的描述，是对数据模型中的“型”的描述，不涉及具体的“值”。模式的一个具体值称为该模式的一个实例。数据库系统具有外模式、模式和内模式三级模式结构，通过“外模式/模式”和“模式/内模式”二级映像提供数据的独立性。



习题 2

1. 设一个计算机系统的硬件组成如图 2.1 所示，在一般情况下，当用户在键盘上输入一串字符时，试问该串字符将被送往哪些硬件组件中？

2. 设一个应用程序具有显示字符串“北京欢迎您！”的功能，在该应用程序运行过

程中，执行显示功能前，该字符串存放在什么硬件组件中？执行显示功能时，该字符串将被送往哪些硬件组件中？

3. 在计算机中引入高速缓冲存储器的目的是_____，在计算机结构中，以下哪个说法正确？

A. 高速缓存介于主存与硬盘之间　　B. 高速缓存介于 CPU 与主存之间

C. 高速缓存介于键盘与主存之间　　D. 高速缓存介于主存与显示器之间

4. 什么是进程？什么是线程？进程与线程之间是一种什么样的关系？

5. 已知某计算机的内存容量为 2GB，该计算机上运行 Linux 操作系统，假设在某一时刻，系统中共有 30 个用户进程在运行，试问每个用户进程可以分配到多少虚拟地址空间？

6. 计算机的硬件组成大体上可以划分为哪几个子系统？这些子系统分别提供什么功能？它们是借助什么进行连接进而构成完整的计算机系统的？

7. 设一台计算机的内存大小为 2GB，试问需要多少比特（位）的地址来寻址内存中的任意一个字节？（提示：B 表示字节，1K 为 1024，1M 为 1KK，1G 为 1KM。）

8. 在计算机中，什么是 RAM？什么是 ROM？它们分别有什么主要特点？分别用于什么用途？试举例说明。

9. 什么是操作系统？有操作系统的计算机和没有操作系统的计算机主要有什么不同？

10. 一个操作系统通常由哪些主要部分构成？Linux 操作系统和 Windows 操作系统中的哪些组件分别与这些组成部分相对应？

11. 在操作系统体系结构中，什么是单体内核结构？什么是微内核结构？Linux 操作系统和 Windows 操作系统分别是什么样的结构？

12. 支持对称多处理技术的操作系统与只支持单处理器的操作系统在内部处理能力方面有什么不同？从用户使用的角度看有什么不同？

13. 数据库系统与数据模型之间是什么样的关系？试举例加以说明。

14. 数据模型包括概念模型、逻辑模型和物理模型，它们的作用分别是什么？当我们判断一个数据库系统属于关系数据库系统还是属于面向对象数据库系统时，主要是依据哪一种模型来判断的？

15. 用于表示概念模型的最著名方法是什么方法？它是如何表示概念模型的？

16. 设某关系数据库中的一张表如图 2.16 所示，试说明该表包含哪些属性？拥有哪些元组？试给出用 SQL 描述从该表中查询所有女生的学号和姓名的方法。

17. 什么是数据库模式？试举例加以说明。

18. 什么是外模式、模式和内模式？采用三级模式有什么意义？

第 3 章
可信计算平台基础



安全与可信密不可分。国际可信计算组织（TCG，Trusted Computing Group）提出的可信计算技术规范与我们要介绍的信息系统安全的内容有密切的关系。TCG 的可信计算体系以硬件安全芯片为信息系统安全的硬件基础，构建可信计算平台，通过建立从硬件层到应用软件层的信任链来解决信息系统的可信问题。本章以 TCG 的可信计算体系为代表，介绍可信计算平台的主要思想与基本技术，为从硬件与软件相结合的整体角度解决信息系统的安全问题奠定良好的基础。

3.1　可信计算发展概貌

早在 20 世纪 70 年代末，尼巴尔第（G.H. Nibaldi）就对可信计算的概念进行了探讨，建立了可信计算基（TCB，Trusted Computing Base）的思想。该思想为美国国防部的 TCSEC 标准的制订奠定了重要的基础。可信计算基思想的重要启示之一是通过硬件、固件和软件的合作来构筑系统平台的安全性和可信性。

可信计算要研究的根本问题是信任问题。信任问题的本质是实体行为的可预测性和可控制性，即实体的完整性。因此，如何度量和维护实体的完整性自然是可信计算的关键。

软件与硬件结合是实现完整性度量的重要途径，以可信平台模块（TPM，Trusted Platform Module）为基础的可信计算技术是这方面的代表性成果之一。

定义 3.1　可信平台模块（TPM，Trusted Platform Module）是 TCPA/TCG 定义的可信计算平台的核心基础部件，通常以硬件的形态存在，提供系统完整性度量功能及完整性度量信息的存储和报告功能，同时提供密码运算、密钥生成和管理、数据封装等功能。

1999 年，可信计算平台联盟（TCPA，Trusted Computing Platform Alliance）的创立是该技术发展的重要推动因素。起初，TCPA 由微软、英特尔（Intel）、IBM 等 190 家公司参加，它致力于数据安全的可信计算，包括研制密码芯片、特殊的 CPU、主板或操作系统安全内核。

2003 年 4 月，TCPA 演变为可信计算组织（TCG，Trusted Computing Group），它在 TCPA 强调安全硬件平台构建的宗旨之外，更进一步地增加了对软件安全性的关注，旨在从跨平台和操作环境的硬件组件和软件接口两方面，促进不依赖特定厂商的可信计算平台工作标准的制订。

3.1.1　TCG可信计算的典型前期基础

TCPA 和 TCG 把可信计算向实用化推进了一大步，而业界在它们成立之前开展的很多研究工作，则为 TCPA 和 TCG 的技术体系的建立打下了重要的基础。以下几项工作具有典型的意义。

1991 年，卡内基梅隆大学的泰格（J.D.Tygar）等人提出了基于安全协处理器的 Dyad 系统模型。该模型的安全协处理器硬件为系统提供私密性和完整性支持。它通过数字签名检验操作系统和其他系统软件的完整性，为进入运行状态的操作系统提供完整性验证和加解密等服务，并支持操作系统验证其他组件的完整性、实现信息的加解密、建立与远程系统的加密连接。

1994 年，美国可信信息系统公司的克拉克（P.C. Clark）和乔治华盛顿大学的霍夫曼（L.J. Hoffman）给出了基于智能卡的引导完整性令牌系统（BITS，Boot Integrity Token System）模型。该模型把系统的主引导程序存放在智能卡中，以保护主引导程序的完整性。存放在智能卡中的还有其他引导文件的哈希值及用户口令和主机标识。系统启动时，首先验证用户使用智能卡的合法性和智能卡与主机的匹配关系，然后从智能卡中读取主引导程序，开始引导过程。主引导程序从主机中读取其他引导文件，完成引导过程。存放在主机中的文件的完整性借助智能卡中的哈希值进行验证。

1997 年，宾夕法尼亚大学的阿玻（W.A. Arbaugh）等人提出了 AEGIS 安全引导体系结构模型。该模型修改了主机系统的 BIOS，并增加一个 AEGIS ROM，以实现对可执行代码的完整性检查，如果完整性检查失败，则提供系统恢复支持。该模型把引导过程涉及的系统组件划分为 6 层，第 0 层是基础 BIOS 和 AEGIS ROM，包含验证代码、公钥证书和系统恢复代码；第 1 层是其余的 BIOS；第 2 层是扩充的只读存储；第 3 层是操作系统的引导块；第 4 层是操作系统；第 5 层是应用软件。第 0 层中的软件是可信软件，用做完整性检查链的根。其余各层的可执行代码在执行之前，由低层进行完整性检查。完整性检查通过哈希值和数字签名实现。

2001 年，IBM 沃森研究中心的代尔（J.G. Dyer）、达特茅斯学院的史密斯（S.W. Smith）和美国 Cryptographic Appliances 公司的魏因加特（S. Weingart）等人研制了 IBM 4758 安全协处理器。它在一个物理装置中封装了三组成分：硬件、固件和软件。硬件通过 PCI 接口与主机系统连接；固件包含 POST 和微引导程序；软件包含操作系统装载程序、操作系统和应用软件。IBM 4758 是一个独立的缩微计算机系统，支持应用软件在其内部运行，安全应用软件可以部署在其内部，并通过安全协议与主机系统中的软件通信，构成大的应用系统。IBM 4758 内部把组件划分为若干层，借助数字签名实现内部系统的安全引导。

实际上，对 IBM 4758 的研究工作远在 1999 年 TCPA 成立以前就已经启动了。以上这些，都是可信计算相关的具有代表性的系统方面的研究工作，它们对 TCG 的可信计算技术体系的建立具有重要的意义。

3.1.2　TCG可信计算的发展思路

TCG 继承了 TCPA 的技术规范，它的目标是为计算机系统平台的可信性提供强有力的支持，基本方法是在现有计算机系统的基础上增加 TPM 功能模块，把现有计算机系

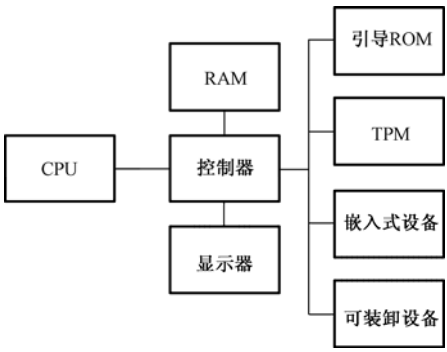


图 3.1　基于 TPM 的可信计算平台参考模型

统平台扩充为可信计算平台，相应的可信计算平台参考模型如图 3.1 所示。

TCG 从一开始就致力于可信终端及相关技术的应用，其规范为未来的可信计算勾勒出一个大体的框架，该框架的重要思想之一是通过增强现有的终端体系结构的安全性来保证整个系统的安全。其主要思路是在各种终端（包含 PC、手机及其他移动智能终端等）硬件平台上引入可信架构，通过其提供的安全特性来提高终端系统的安全性。

终端可信的核心是 TPM 可信芯片。以 TPM 为基础的可信计算蕴涵着以下几个方面的思想：

- 用户的身份认证是对使用者的信任；
- 平台的完整性体现使用者对平台运行环境的信任；
- 应用程序的完整性体现应用程序运行的可信性；
- 平台之间的可验证性体现网络环境中终端之间的相互信任。

3.1.3　响应TCG规范的热点研究

TCPA 和 TCG 的可信计算体系规范的推出，引起了工业界和学术界的广泛关注，大量的研究工作和研究成果不断涌现。

2004 年，IBM 公司的丸山（H. Maruyama）等人在他们设计实现的 TPod 体系结构中利用可信计算平台实现了系统的可信引导。在 TPod 实现的可信引导中，基础 BIOS 作为信任根首先执行并度量其余 BIOS 的完整性，其余 BIOS 执行并度量操作系统装载程序 GRUB 的完整性，GRUB 度量操作系统（这里包括 SE-Linux 内核和/etc/init 脚本等）的完整性。

在完整性度量中，把操作系统等作为简单组件对待属于粗粒度的问题处理方法。从操作系统层开始，直到应用软件层，若把度量对象的粒度细化到与实际应用系统比较一致的程度，则需要解决更多的问题。细粒度的完整性度量是当前的热点研究问题。

2004 年，IBM 沃森研究中心的塞勒（R. Sailer）等人提出了完整性度量的 IMA（Integrity Measurement Architecture），设计实现了 Linux 系统的组件细化完整性度量原型系统。IMA 原型以可信计算平台为可信硬件，给出了系统 BIOS、操作系统装载程序 GRUB、Linux 内核、Linux 模块、应用软件等各层组件执行前的完整性度量方法，重点给出了 Linux 内核、Linux 可装载内核模块、动态可装载库、结构化数据和可执行用户程序等的度量方法。

2005 年，卡内基梅隆大学的史（E. Shi）和 IBM 沃森研究中心的范·多恩（L.Van Doorn）等人提出了为分布式系统建立可信环境的 BIND（Binding Instructions aNd Data）框架。BIND 把代码的完整性证明细化为关键代码段的完整性证明，并为关键代码段产生的每一组数据生成一个认证器。认证器附着到相应的数据上，从而实现关键代码段的完整性证明与其所产生的输出数据的绑定。因此，BIND 可以通过关键代码段及其输入数据的完整性证明来达到系统完整性证明的目的。

2006 年，宾夕法尼亚州立大学的耶格（T. Jaeger）、IBM 沃森研究中心的塞勒和加州大学伯克利分校的山克（U. Shankar）提出了基于信息流的 PRIMA（Policy-Reduced Integrity Measurement Architecture）完整性度量体系结构，并研究了以 SE-Linux 为基础的原型系统。PRIMA 项目的研究工作在 IMA 研究成果的基础上引入 CW-Lite 信息流模型来处理组件依赖关系，在基于信息流的系统完整性动态度量方面进行了卓有成效的尝试。

在开放式、分布式的全球计算环境中，建立信任关系是现实应用的迫切需求。可信计算受到了工业界和学术界的高度重视，很多成果已经进入我们的视野。然而，在可信信息系统建设的道路上，依然“路漫漫其修远兮”，尚需大家“上下而求索”。

3.2　可信计算平台的基本特性

在 TCG 的定义中，可信是对一个实体为了实现特定的目标而按照特定的方式进行工作的一种期望。这个定义将可信计算和当前的安全技术分开：可信强调行为可预期，但并不等于行为是安全的，这是两个不同的概念。

根据英特尔公司的密码与信息安全专家大卫·格劳洛克（David Grawrock）的说法，如果你知道自己的计算机中有病毒，知道这些病毒会在什么时候发作，了解发作后会产生怎样的后果，同时病毒也确实是这么运行的，那么这台计算机就是可信的。

一个可信计算平台要达到可信的目标，最基本的原则就是必须真实报告系统的状态，同时，决不暴露秘密信息，尽量不表露自己的身份。这就需要三个必要的基础特征：

- 保护能力（Protected Capabilities）；
- 对外证明（Attestation）；
- 完整性度量、存储和报告。

3.2.1　保护能力

保护能力是一组具有对保护区域进行访问的特定权限的命令，保护区域指的是可在其中安全地操作敏感数据的地方（如内存、寄存器等），是只有保护能力才能访问的数据区域。

TPM 实现保护能力和用于保护和报告完整性度量信息的保护区域，这样的保护区域通常体现为平台配置寄存器（PCR，Platform Configuration Register）。

TPM 也存储用于对报告提供的度量结果信息进行认证的密钥。

此外，TPM 的保护能力还可以包括其他安全功能，如密钥管理、随机数生成、用系统状态封装数据等。

3.2.2　对外证明

定义 3.2　对外证明是证明实体完整性相关性质的过程。

外部实体可以对保护区域、保护能力和信任根进行证明。平台可以对影响平台完整性的平台特性的描述进行证明。所有的对外证明都要求证明方提供可靠的证据。

对外证明具有多种不同层面的含义：由 TPM 提供的证明、对平台进行的证明、由平台进行的证明和平台的认证。

由 TPM 提供的证明指的是这样一种操作，该操作为 TPM 所掌握的数据提供证明，实际上，就是使用 TPM 的工作身份密钥（AIK，Attestation Identity Key）对 TPM 的特定内部数据进行数字签名。完整性度量结果和 AIK 本身的可接受性和有效性由某个证明方确定。AIK 可以借助 TPM 的真实身份密钥（EK，Endorsement Key）来生成。

对平台进行的证明指的是这样一种操作，该操作证明平台能够可信地报告完整性度量结果，该操作的实施需要使用平台的相关凭证（Credential）集合或该集合的子集，该操作用于发放 AIK 凭证。

由平台进行的证明指的是这样一种操作，该操作为平台的一组完整性度量结果提供证明，实际上，就是用 TPM 的 AIK 对一组 PCR 进行数字签名。

平台的认证指的是为平台的身份提供证据，该身份可以与某个用户或其行为有关，也可以无关。平台的认证使用的是不可迁移的签名密钥。已被证明的密钥（即由 AIK 签名的密钥）拥有“可以被证明”的附加语义。由于有无数的不可迁移密钥与一个 TPM 关联，所以，可以被认证的身份是无数的。

3.2.3　完整性度量、存储和报告

定义 3.3　完整性度量指的是获取影响平台完整性（可信性）的平台特性的度量值并把该度量值的摘要存放到 PCR 寄存器中的过程。

度量的起始点称为度量的信任根。度量的静态信任根从一个众所周知的起始状态（如上电自检状态）开始度量，度量的动态信任根从一个非可信状态转移到一个可信状态。

定义 3.4　完整性存储指的是存储完整性的度量值，以备后用，度量值通常存储在 PCR 寄存器中。

定义 3.5　完整性报告指的是对外证明记录在 PCR 寄存器中的完整性度量结果的

过程。

完整性存储是介于完整性度量与完整性报告之间的步骤。应该进行完整性存储，不然，为了解释 PCR 寄存器的值，将需要再次进行完整性度量。

完整性度量、存储和报告的理念是：可以允许平台进入任何可能的状态，包括不受欢迎的状态和不安全的状态，但是，不允许平台谎报它所处的状态。可以通过独立的过程来评估完整性状态并确定合理的响应。

3.3　可信计算平台的基本体系

在 TCG 的技术体系中，信任根是默认的可以信任的组件，它们总能正确地工作，无须外部监督。信任根的完整集合必须至少具有描述影响平台可信性的平台特征所需要的最小功能集。

一个可信计算平台通常拥有三个信任根：度量的信任根（RTM，Root of Trust for Measurement）、存储的信任根（RTS，Root of Trust for Storage）和报告的信任根（RTR，Root of Trust for Reporting）。RTM 是本质上能够实施可靠的完整性度量的计算引擎，在典型情况下，它是由度量的核心信任根（CRTM，Core Root of Trust for Measurement）控制的常规的平台计算引擎。CRTM 是由以 RTM 形态工作的平台执行的指令。RTM 也是传递性信任链的根。RTS 是能够维护完整性摘要及摘要序列的总貌的计算引擎。RTR 是能够可靠地报告 RTS 所持有的信息的计算引擎。

3.3.1　平台的可信构件块

定义 3.6　可信构件块（TBB，Trusted Building Block）是信任根中没有保护区域或保护能力的成分，通常，它包括 RTM 指令和 TPM 初始化功能，一般是平台相关的。

例 3.1　以基于 TPM 的计算机平台参考模型为基础，试说明什么样的成分可以构成一个 TBB。

解答：这些成分可以构成一个 TBB：CRTM、CRTM 存储与主板的连接、TPM 与主板的连接、键盘等，可信计算平台中的可信构件块如图 3.2 中的粗线所示。

我们说 TBB 是可信的，其中的含义是：我们可以预期它不会做出破坏可信计算平台的目标的行为。

3.3.2　信任边界

TBB 和信任根的组合可以形成一个信任边界，在该信任边界内，可以实现对一个平台的最小配置环境的完整性度量、存储和报告。对较复杂的系统的度量，则除了 CRTM

之外，还需要借助其他的 ROM 代码，为此，系统在把执行的控制权传递给该 ROM 代码之前，需要对其进行度量。建立 TBB 时，要确保含有其他度量代码的组件不会不经意地扩展信任边界，只有确立了被扩充成分的可信性之后，才会出现信任边界扩展的行为。

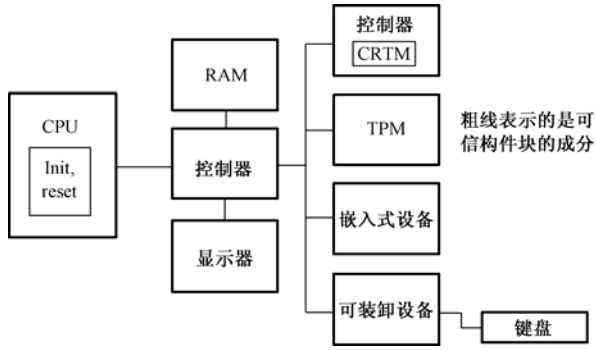


图 3.2　可信计算平台中的可信构件块

3.3.3　信任传递

信任传递也称信任推导，是信任从信任根向外传递的过程。首先，信任根描述第二组功能的可信性，根据这个描述，某相关的实体可以确定该组功能的可信性，如果该实体断定该组功能的可信程度是可以接受的，那么，信任边界从信任根扩展到包含第二组功能的范围。这样的过程可以不断重复下去，第二组功能可以描述第三组功能的可信性，依此类推。信任传递用于描述平台特征的可信性，并证明不可迁移的密钥是不会迁移的。

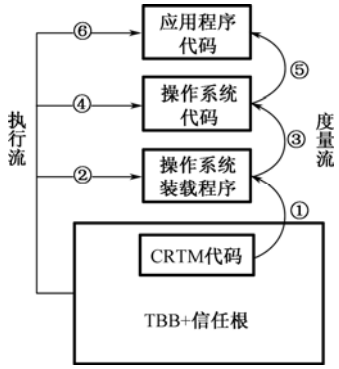


图 3.3　系统引导过程中的信任传递

图 3.3 给出了系统引导过程中的信任传递情况。起初，信任边界仅包含静态信任根，随后，把信任根之外的代码逐渐纳入进来，扩展信任边界。在信任边界的每一次扩展中，首先度量目标代码，然后再把控制权传递给它。

3.3.4　完整性度量

TCG 的技术体系使用度量核、度量事件、被度量的值和度量产生的摘要等概念描述完整性度量行为。度量核产生度量事件，度量事件由被度量的值和度量产生的摘要等两类数据构成，被度量的值是程序代码或内嵌数据的表示，度量产生的摘要是被度量的值的哈希值。度量核对数据进行扫描并产生消息摘要，摘要是平台的操作状态的快照。两类数据的存储是相互隔离的，度量产生的数据通过 RTR 和 RTS 功能存储在 TPM 中，被

度量的值理论上可以存储在任何地方，由度量核确定，实际上，也可以不存储该值，可以在需要的时候再重新计算生成。

度量数据描述被度量组件的属性和特征。被度量的数据字段由度量核的实现者负责进行详细的解释，以产生符合度量事件的用户之需的编码结果。实现者负责确定如何划分事件数据。平台相关的 TCG 规范进一步说明平台的配置、平台的表示和需要对平台进行度量的预期用户。

储存的度量日志（SML，Stored Measurement Log）包含有关的被度量的值的序列，每个序列共享一个公共的度量产生的摘要。在度量中，把被度量的值附加到该公共摘要上，然后再重新计算形成结果的哈希值，这种操作通常称为摘要的拓展，拓展操作确保所有被度量的值都不被忽略，而且，使度量的次序得到维持。

TPM 用一组 PCR 寄存器来保存度量产生的摘要，PCR 寄存器的值不是永久的，它们在系统引导时复位。PCR 寄存器的更新方法由以下代数式表示：

$$PCR[n] \leftarrow SHA1\ (\ PCR[n] + M_data\)$$

其中的 M_data 表示被度量的数据。

验证度量事件时，需要重新计算度量产生的摘要，并把它与 PCR 的值进行简单的比较。TCG 没有给 SML 内容定义数据编码规则，它推荐遵循诸如 XML 的适当标准，以确保广泛的可用性。不同的平台可以要求不同的内容表示，所以，平台相关的规范（如个人计算机相关的平台规范）可以定义其特定的编码规则。

SML 可能会变得非常大，因而，它不适合存放在 TPM 中。对 SML 的攻击可以容易地检测出来，所以，SML 不需要由 TPM 提供保护。不过，SML 还会受到拒绝服务的攻击，实现者应该采取措施，复制或重新生成该日志。

3.3.5　完整性报告

RTR 有两个功能：一是展示用于存储完整性度量信息的保护区域；二是根据可信平台的身份证明储存的数据的真实性。PCR 可以用易失性或非易失性存储器来实现，它们必须能够抗击软件攻击，同时，也应考虑防止物理篡改攻击。可以采用 AIK 对完整性报告进行数字签名，以认证 PCR 的值的真实性，还可以把一个随机数组合到 PCR 的签名中，以防止重放攻击。

AIK 由 TPM 生成和管理，一个 TPM 可以拥有很多 AIK。平台的属主可以采用一个特定的 AIK 来保护私密性，以防止合谋攻击。

TPM 的生产厂家可以在出厂前为每个 TPM 生成唯一的 EK，以便标识 TPM 的真实身份。EK 用于为 TPM 发放 AIK 凭证，以及确定平台的属主。平台的属主可以创建存储根密钥，用于对 TPM 的其他密钥进行加密。

1．完整性报告协议

完整性报告可用于确定平台的当前配置。用于报告完整性度量情况的对外证明协议与消息交换如图 3.4 所示。

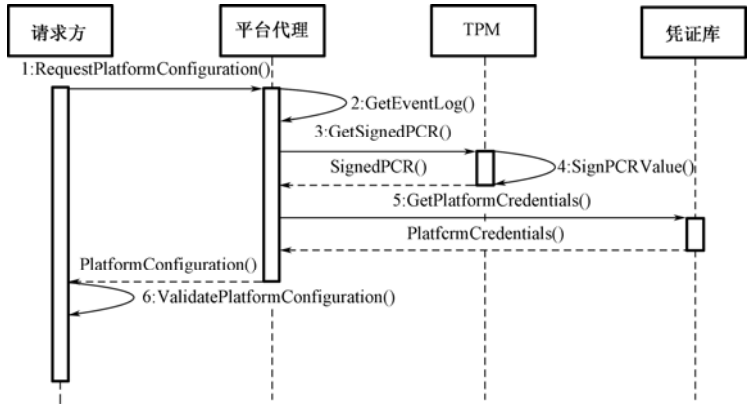


图 3.4　对外证明协议与消息交换

报告完整性的度量情况就是对外证明平台的配置情况，相关协议由以下步骤构成：

（1）证明的请求方向平台发出获取一个或多个 PCR 寄存器值的请求；

（2）配备有 TPM 的平台上的度量机制采集 SML 记录信息；

（3）度量机制从 TPM 中获取 PCR 寄存器的值；

（4）TPM 用 AIK 对 PCR 寄存器的值进行签名；

（5）平台的度量机制采集与 TPM 关联的凭证，并把 SML 记录信息、凭证和经过签名的 PCR 寄存器的值提供给证明的请求方；

（6）证明的请求方验证请求的响应结果，计算度量产生的摘要，将其与 PCR 寄存器的值进行对比，并评估平台的凭证，检查签名信息。

该协议独立于信息传输或发布机制，可以采用现有的通信、消息和远程过程调用基础设施传输对外证明信息。

2．平台的身份与隐私

对外证明的目的之一是使证明的请求方能够确定某个 TPM 给一个消息签了名，它也能够确定哪一个 TPM 给该消息签了名。利用私密性证书机构给 AIK 签发凭证，可以在不向证明的请求方泄露 EK 的秘密信息的前提下确保平台的可信性。TPM 向私密性证书机构提供 AIK 的公钥信息，由私密性证书机构为该 AIK 发放凭证。AIK 凭证的合法使用要求 TPM 证明相应的 AIK 是唯一地绑定到该 TPM 上的，通过使用 TPM 中的 EK 的私钥对 AIK 凭证进行解密的方法可以做到这一点，只有拥有正确的 EK 的私钥的 TPM 才有可能成功地实施这样的解密操作。

私密性证书机构是可信的机构，它不会泄露平台的敏感信息，包括 EK 的公钥信息或由其导出的可以标识个体身份的信息；它也不会错误地表达其发放的 AIK 凭证所对应

的平台的信任属性。

通过对 TPM 进行配置，可以要求实体在开始执行 AIK 凭证发放协议前提供属主授权证明。为了支持对 TPM 的使用控制，TPM 提供关停机制，允许使用者关闭其功能。TPM 的功能被关闭后，该 TPM 在平台中就不起作用了。

3．平台的对外证明

平台的对外证明就是让平台以外的实体能够验证平台的可信性，所要做的工作是对相关的证据进行评价，以确定能否相信给定的 AIK 可用于实现对指定的平台的认证。为了验证某个特定的 AIK 所关联的信任声明的有效性，需要使用由 TPM 和平台的生产厂家发放的凭证。

4．平台中使用的凭证

TCG 定义了五种类型的凭证，每种凭证只为执行一种特定的操作提供所需要的信息，这五种凭证类型分别是：真实身份凭证（即 EK 凭证）、符合性凭证、平台凭证、有效性凭证、工作身份凭证（即 AIK 凭证）。

1）真实身份凭证

真实身份凭证由生成 EK 的实体发放。EK 的生成属于 TPM 生产过程中的工作，一般来说，EK 是由 TPM 的生产厂家生成的。为了使 TPM 的生产厂家能够声明已经正确地创建了 EK 并把它嵌到了有效的 TPM 中，应该在把平台发行给用户之前生成 EK。

如果在把平台发行到用户手中之后才生成 EK 密钥对，生成密钥时的条件会对它所提供的平台身份的真实性有一定的影响。

真实身份凭证包含以下信息：

- TPM 生产厂家的名称；
- TPM 部件模型编号；
- TPM 版本信息；
- TPM 的 EK 公钥。

从隐私的角度上考虑，EK 公钥也属于敏感信息，因为它唯一地标识了一个特定 TPM 的身份，进而也唯一地标识了相应特定平台的身份。每个 TPM 都需要一个 EK 凭证。

2）符合性凭证

符合性凭证可以由任何有资格对 TPM 或包含 TPM 的平台进行评价的实体发放，此类实体可以是平台的制造商、销售商或独立的第三方。符合性凭证表明评价者认为 TBB 的设计与实现与公认的评价基准是相符的。通过对凭证进行签名，评价者提供了对评价结果的证明，评价结果的细节可供审查用。为了支持评价工作，TCG 定义了内容丰富的评价标准和准则。

可以给一个平台发放多个符合性凭证，其中一个与 TPM 相对应，其余的与 TBB 的不同组件相对应。

符合性凭证可以包含以下信息：

- 评价者名称；
- 平台制造商名称；
- 平台模型编号；
- 平台版本；
- TPM 制造商名称；
- TPM 模型编号；
- TPM 版本。

符合性凭证不包含唯一地标识平台的信息。

在每个平台模型中可以存在几个符合性凭证，但相同生产批次和模型的多个平台只需要一套凭证。

3）平台凭证

平台凭证可以由平台制造商、销售商或有资格的任何实体发放，它标识平台的制造商，并描述平台的特性。它也引用与 TPM 关联的平台真实身份凭证和有关的符合性凭证。凭证的引用由被引用的凭证的消息摘要构成。我们可以认为平台凭证是对隐私敏感的，因为它与多个同类平台中的某个特定平台相关联。

平台凭证包含以下信息：

- 平台制造商名称；
- 平台模型编号；
- 平台版本；
- 真实身份凭证；
- 符合性凭证。

平台凭证提供证据以证明该平台包含真实身份凭证所描述的 TPM。每个平台可以拥有一个平台凭证。

可度量的硬件或软件组件的制造商可以为组件提供参考度量值，参考度量值是在制造期间当组件处于正常工作秩序时计算出的被度量的组件的摘要。典型地，计算是在功能性测试之后进行的。并非所有的组件都需要建立参考度量值，只有必要查检那些可能给安全带来威胁的组件，以防止被注入后门。

4）有效性凭证

通常，期望的摘要的使用者信任净室操作员的声明，所以，可以在任何时候进行净室度量。可以通过把参考度量值与运行时的实际度量值进行比较来检测组件可能发生的变化。经过签名的描述拥有预期的摘要的组件结构的文档称为有效性凭证。

以下是可以拥有有效性凭证的一些组件实例：

- 视频适配器；
- 磁盘适配器；

- 内存控制器；
- 通信控制器/网络适配器；
- 处理器；
- 键盘和鼠标；
- 软件。

有效性凭证由有效性实体发放，愿意并且能够度量和证明被度量的值的任何人都可以称为有效性实体。通常，组件的制造商最能够生成期望的值。组件描述的任何部分都可以是信任判定的素材，不过，有效性凭证的候选组件很有可能是给安全带来威胁的那些组件。

组件描述至少包括以下元素：

- 有效性实体名称；
- 组件制造商名称；
- 组件模型编号；
- 组件版本；
- 度量产生的值。

组件描述可以包含组件的能力。

一个模型、一系列组件或被签名的信息的唯一性都可以确定一个有效性凭证，有效性凭证可以由实施有效性验证的实体以任何合适的方式分发和公布，最好是以电子方式公布，以便能由自动化工具进行处理。

想要证明事件数据的准确性的实体可以将 PCR 寄存器的值与有效性凭证中的度量生成的值进行对比。

拥有有效性凭证的组件有时需要进行改错、更新或升级，这会导致包含在原始有效性凭证中的预期的结果失效。可以根据平台相关的规范处理组件与有效性凭证间的一致性问题。

5）工作身份凭证

标识用于给 PCR 寄存器的值签名的 AIK 私钥，它包含 AIK 公钥和发放者认为有用的任何其他信息，由能够验证各种凭证和保护客户隐私的实体发放。

通过发放工作身份凭证，签名者通过证明 TPM 的实情而对外证明 TPM 的真实性，该证明的目标是证明该 TPM 拥有该 AIK 且该 AIK 与有效的真实身份凭证、平台凭证和符合性凭证联系在一起。可信第三方进一步确保该签名者遵守客户对隐私的期望，该期望可能包括保护可标识的个体信息。

工作身份凭证按以下方式引用其他凭证。

（1）工作身份凭证包含对 TPM 制造商和模型的引用，但不包含对隐私敏感的 EK 的引用。

（2）工作身份凭证也包含对平台制造商和模型的应用，不过，那不是对平台凭证本

身的引用，而是对包含在平台凭证中的隐私不敏感的信息的引用。

（3）工作身份凭证包含一个指向证明的请求方能够找到 TPM 和平台的符合性文档的地方的指针。

借助对外证明协议，证明的请求方可以使用这些信息和凭证中的其他信息来建立对平台的信任感。

图 3.5 描述了平台中使用的各种凭证及它们之间的关系。

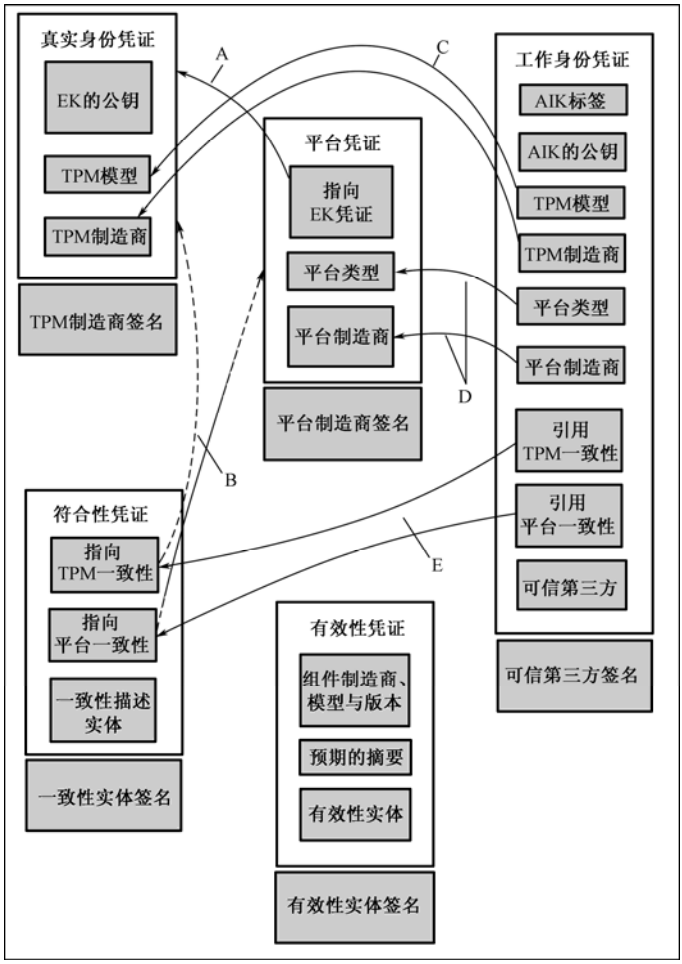


图 3.5　可信计算平台中的凭证

描述凭证时需要说明隐私敏感性问题。以上介绍的每一个凭证都是建立平台的可信性所必需的，平台的制造商必须为这些凭证提供保护，以保护用户的隐私。

3.3.6　以TPM为通信端点

考虑系统间的信息交换时，安全的分布式系统的设计者必须标识通信端点的身份。

对于系统的整体安全性而言，通信端点的组成和结构与通信协议一样重要。通信端点至少应提供非对称密钥、密钥存储和协议数据项的保护处理。

基于非对称密码技术的经典消息交换方法用消息接收者的公钥对消息进行加密以确保消息接收者的唯一性，用消息发送者的私钥对消息进行签名以确保消息的防篡改性。密钥存放在通信端点中，没有得到妥善管理的密钥能导致安全性的丧失。另外，没有得到妥善配置的通信端点也能导致安全性的丧失。TPM 通过提供密钥管理和配置管理特性来提高安全性。这些特性的组合可以实现密钥与平台配置的封装，从而进一步强化通信端点的定义。

TCG 定义了四类保护信息交换的操作：绑定（Binding）、签名（Signing）、封装（Sealing）和封装的签名（Sealed-Signing）。

1．绑定

定义 3.7　用一个实体的公钥对一项信息进行加密的过程称为信息与实体的绑定。通常，绑定的情形是信息的发送方用信息的接收方的公钥对信息进行加密。

只有使用信息接收方的私钥才能对加密的信息进行解密，以恢复信息的原状。当私钥由 TPM 当做不可迁移的密钥进行管理时，只有创建该密钥的 TPM 才能使用它。这样，用该公钥加密的信息便被绑定到一个特定的 TPM 上。

创建可以在多个 TPM 间传递的可迁移的私钥是有可能的，在这种情况下，除了加密之外，绑定就没有其他特殊的含义了。

2．签名

在传统意义上，签名把信息的完整性与用于产生签名的密钥关联在一起。TPM 把一些密钥标记为只能用于签名的密钥，也就是说，这些密钥只用于计算被签名的数据的哈希值，并对该哈希值进行加密，因此，不能把它们误用为加密密钥。

3．封装

定义 3.8　把一项信息与一个平台的某个状态绑定在一起的过程称为信息在平台上的封装，平台的状态通常由平台的完整性度量值表示。

封装比绑定更进了一步，它把信息与信息的发送者指定的一组平台度量值绑定在一起，平台度量值指定了允许对信息进行解封装前平台必须呈现的配置状态。封装把被加密的信息与一组 PCR 寄存器的值及一个不可迁移的非对称密钥关联在一起，被加密的信息实际上用对称密钥进行加密。

信息的封装方法是：选择一组 PCR 寄存器的值，用非对称加密算法（通过公钥）对该组 PCR 寄存器的值和一个对称密钥进行加密，然后用该对称密钥对待封装的信息进行加密。只有当平台的配置与信息的发送者指定的 PCR 寄存器的值相匹配时，拥有

非对称解密密钥（即私钥）的 TPM 才对相应的对称密钥进行解密。

封装是 TPM 的一个强大的特性，它确保只有当平台在一个非常明确的已知配置下工作时才能通过解密使加密保护的信息恢复原状。

4．封装的签名

签名操作也可以与 PCR 寄存器关联在一起，以进一步确保对信息进行签名的平台满足特定配置的要求。验证方要求一个签名必须包括一组特定的 PCR 寄存器。在签名操作期间，签名方采集特定寄存器的值，把它们组合到给定的信息之中，在计算待签名的信息的摘要时，它们作为计算的输入信息的一部分，影响计算的输出结果。验证方可以检查签名信息中提供的 PCR 寄存器的值，这相当于检查在创建信息的签名的那一刻执行签名操作的平台的配置。

3.3.7　存储保护

RTS 保护托付给 TPM 的密钥和数据的安全，它管理少量的易失性存储器，用于在执行签名和解密操作时存放密钥。可以对不活跃的密钥进行加密，并把它们转移到 TPM 之外，从而为其他更活跃的密钥腾出空间。位于 TPM 外部的密钥缓存管理器（Key Cache Manager，KCM）负责管理密钥槽的高速缓存，KCM 通过接口与可以长期存放不活跃的密钥的存储设备进行交互。一个通用的受保护的存储服务也允许保存不透明的数据，在这种情况下，RTS 的数量将翻倍。

经过优化，RTS 可以存放小的对象，其大小约与一个非对称密钥减去开销后相当，其中的开销是 210 字节左右的有效载荷，可以存放的对象有多种类型，如对称密钥、非对称密钥、口令字符串、网页附带的小信息段、认证结果及不透明的数据。

对 TPM 而言，有三种类型的密钥并非是不透明的，它们是 AIK 密钥、签名密钥和存储密钥。存储的根密钥（SRK）和真实身份密钥（EK）是两种内嵌在 TPM 中的密钥，如图 3.6 所示，它们是不能移到 TPM 的外部的。为平台创建一个新属主时，可以附带地创建一个新的 SRK，其附带作用就是使得由以前的 SRK 控制的所有数据对象都处于被加密的状态。

SRK 是由 TPM 生成的，建立 TPM 的属主时，系统采用 EK 对 SRK 的口令字符串进行加密。SRK 用于保护存放在 TPM 的外部的其他密钥。

1．密钥属性

RTS 管理的所有密钥都有一个迁移属性，该属性表明密钥是否可迁移，它确定一个密钥是否可以从一个 TPM 传送到另一个 TPM。属性的值是在密钥创建时建立的，是不可改变的。在语义上，一个不可迁移的密钥永远与一个特定的 TPM 关联在一起。对一个不可迁移的密钥进行迁移所产生的效果就是允许一个平台冒充另一个平台。AIK 密钥是永远不应迁移的密钥的例子，也就是说，AIK 密钥是不可迁移的密钥。

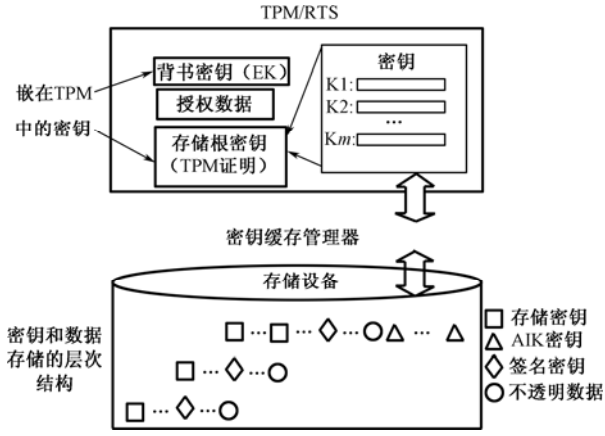


图 3.6　根密钥和真实身份密钥

可迁移的密钥可以在不同的 TPM 之间进行交换，这使得密钥对可以跟着用户走，而不管用户具体使用的是哪一个设备。即使更换了计算平台，在个体间交换的信息依然是可以访问的。

密钥属性不能应用到不透明的数据之上，为了把不可迁移特性扩展到不透明的数据中，应该采用一个不可迁移的密钥把该数据与 RTS 存放在一起。不透明的数据只要受到 TPM 的控制，它就不能在其他地方被解密。不过，数据一旦在 TPM 的外部被解密了，很显然，它就可以迁移到任何系统中。

2．密钥类型

TCG 定义了 7 种密钥类型，每种类型都拥有一组界定其用途的限定。TCG 密钥从广义上可以划分为签名密钥或存储密钥，它们可以进一步划分为平台密钥、身份密钥、绑定密钥、一般密钥和传承密钥。对称密钥被独立划分为认证密钥。

7 种密钥类型如下所述。

（1）签名密钥：这是通用的非对称密钥，用于对应用数据和信息进行签名，它们既可以是可迁移的，也可以是不可迁移的，可迁移的密钥可以在不同的 TPM 之间进行导入和导出，TPM 可以对应用数据进行签名，并实施迁移限定。

（2）存储密钥：这是通用的非对称密钥，用于对数据或其他密钥进行加密，它们的主要用途是对在 TPM 的外部进行处理的密钥和数据进行加密。

（3）身份密钥：这是通常所说的 TPM 的工作身份密钥，即 AIK，是不可迁移的签名密钥，主要用于对 TPM 产生的数据（如 TPM 寄存器的值等）进行签名。

（4）背书密钥：这是通常所说的 TPM 的真实身份密钥，即 EK，是平台的不可迁移的解密密钥。在建立平台的属主时，用于对属主的授权数据进行解密；在创建 AIK 时，用于对相关的信息进行解密，它永远不会用于加密或签名。

（5）绑定密钥：这是用于对少量的数据（如对称密钥等）进行加/解密的密钥，在

一个平台上用于加密，则在另一个平台上用于解密。

（6）传承密钥：这是在 TPM 的外部创建的密钥，属于可迁移的密钥，它们导入到 TPM 中后，可在签名和加密操作中使用。

（7）认证密钥：这是对称密钥，用于保护 TPM 参与的传输会话。

3．外部存储器和密钥缓存管理

TPM 的设计目标是作为适用于客户级计算平台的低成本商品化组件，所以，TPM 拥有的运行用的存储器（易失性存储器）和永久存储器（非易失性存储器）是有限的。而在实际应用中，可能需要无限的存储器，因此，需要定义 TPM 的外部存储器和高速缓存管理器。

为了支持无限的密钥和存储区域，RTS 把密钥组成加密的密钥块，以存放到外部存储器中。在 TPM 之外的密钥块是不透明的，可以存放在任何可用的存储设备中（如闪存、磁盘、网络文件服务器等）。密钥块绑定到特定的 TPM 上，还可以封装到特定的平台配置上。

借助句柄或其他适用的引用机制，密钥块通过它的内容的哈希值来引用。引用标识符可以在密钥缓存管理器（KCM，Key Cache Manager）或其他执行存储功能的应用程序之外消除密钥块的歧义。密钥类型和密钥属性等其他信息可以从外部获得。

TPM 对外提供接口，支持外部程序管理 TPM 的有限的存储资源。通过分离密钥缓存功能和密钥使用功能，可以区分管理功能和应用功能。通常，KCM 只关心密钥的缓存，而应用程序只关心密钥的使用。用于保护其他密钥的存储密钥是一种例外情况，KCM 会同时控制存储密钥的缓存和使用。

封装到特定平台配置的密钥当平台没有处于预定的配置中时也可以被装载，这为在不影响密钥的获取的情况下实现平台在不同状态间的切换提供了方便。每次使用密钥时都要检查相应的配置，所以，在密钥装载时无须检查配置情况，也能确保系统的安全性。

由图 3.6 可见，KCM 是处于 TPM 外部的程序，它负责密钥在 TPM 的易失性密钥槽内存与非易失性外部存储设备之间的移动。KCM 记录所有可用的密钥槽，确定什么时候应该把一个密钥从密钥槽中移走，使另一个密钥可以取而代之。TPM 并不预先报告什么时候密钥槽将被占满，或者什么时候应用程序需要使用一个特定的密钥，所以，应用程序可能需要报告什么时候此类事件会发生，另外，KCM 可能需要实现一个 TPM 接口层，以便应用程序可以通过它获得 TPM 服务。

TPM 提供相应的接口，用于为 TPM 与存储设备之间的切换准备密钥，在任何时刻，KCM 都不能以明文形式呈现密钥。

KCM 可以实现一个模型，对包含在 KCM 管理的存储设备中的密钥块进行索引、存储和检索，可能还包括对使用密钥所需要的口令字符串的管理。

3.4　可信平台模块

TPM 是 TCG 制订的可信计算规范的核心，它可以用硬件实现，也可以用软件实现。本节主要介绍用硬件实现的 TPM。

3.4.1　TPM的组件

图 3.7 描述了支持 RTR 和 RTS 功能的一个 TPM 的逻辑结构及其构成组件，其主要成分包括输入/输出组件、非易失性存储器、PCR 寄存器、工作身份密钥、程序代码、随机数生成器、SHA-1 引擎、密钥生成、RSA 引擎、功能开关和执行引擎等。

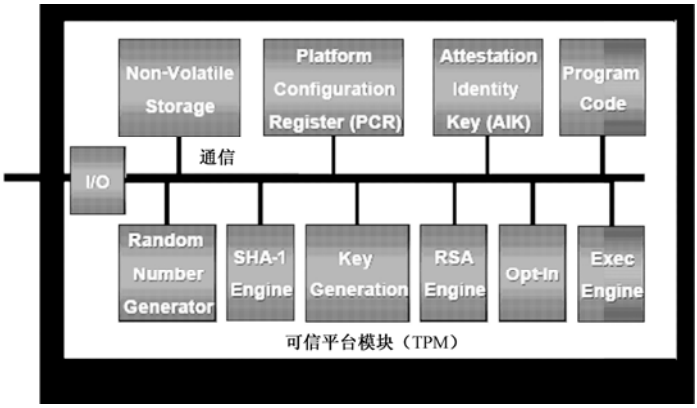


图 3.7　TPM 的逻辑结构及其构成组件

输入/输出组件对通信总线上的信息流进行管理，它执行协议编码和协议解码操作，以适应外部总线与内部总线上的通信需要；确定信息通往所需组件的路由；实施与功能开关组件相关联的访问策略，以及与其他需要访问控制的 TPM 功能相关联的访问策略。

非易失性存储器用于存放真实身份密钥（EK）、存储根密钥（SRK）、属主的授权数据和永久的标记。

PCR 寄存器可以用易失性或非易失性存储器来实现，它们在系统启动或平台掉电时复位。至少需要实现 16 个 PCR 寄存器，其中，第 0～7 号 PCR 寄存器留给 TPM 使用，第 8～15 号 PCR 寄存器可供操作系统和应用软件使用。

工作身份密钥（AIK）必须是永久型密钥，它们应该以经过加密的密钥块的形式存放在外部永久存储器中（在 TPM 之外），而不是永久地存放在 TPM 中的非易失性存储器中。为了提高运行速度，TPM 应该提供足够的空间，以便可以并发地把多个 AIK 密钥块装载到 TPM 的易失性内存中。

程序代码包含对平台设备进行完整性度量的固件。从逻辑上说，这是度量的核心信任根（CRTM），在理想情况下，CRTM 应该包含在 TPM 中，不过，它也可以存在于其

他固件中。

TPM 包含一个真随机比特生成器，可用于作为生成随机数的种子。随机数生成器用于生成密钥、创建即时随机量和增强口令字符串的熵。

SHA-1 消息摘要引擎用于计算签名、创建密钥块及通用用途。

RSA 算法是 TPM 模块中使用的标准算法。RSA 密钥生成引擎用于创建签名密钥和存储密钥。TPM 支持 2048 比特的模数的 RSA 密钥，一些特定的密钥（如 SRK 和 AIK 等）必须拥有至少 2048 比特的模数。

RSA 引擎的作用是使用签名密钥进行签名、使用存储密钥进行加密和解密及使用 EK 进行解密。

功能开关组件的作用是禁用或启用 TPM 模块，还可以是激活或暂停 TPM 模块。功能开关机制包含相应的逻辑和接口，用于确定真实的现场操作状态，以及用于对 TPM 的其他组件实施禁用操作。

执行引擎的作用是运行程序代码，它执行 TPM 初始化操作和完整性度量操作。

3.4.2　通信接口

TPM 的通信接口（即总线结构）是平台相关的，不同的平台可以提供不同的实现，TCG 定义了一种接口序列化转换方法，用于支持各种总线结构或互连方式间的通信。

3.4.3　具有篡改保护能力的装配

TPM 在物理设计上应该具有针对篡改的保护能力，这包括从物理上把 TPM 与平台（如计算机主板）的其他部件绑定在一起，以防止它可被容易地从一个平台上拆卸下来又装到另一个平台上，从而抵抗篡改攻击。

可以采取适当的篡改证据采集措施，以便能够检测通过物理途径实施的篡改攻击。如果 TPM 是用软件实现的，那么，必须证明它具有与硬件实现相当的篡改保护能力，特别地，要证明每个特定的 TPM 都拥有其唯一性标记且都与一个特定的平台绑定在一起。

3.5　可信计算平台的隐私问题

在 TCG 规范确定的可信计算环境中，要借助平台的身份去建立平台的可信性，这涉及敏感的隐私问题。TCG 遵循万维网联盟（W3C，World Wide Web Consortium）的隐私偏好平台（P3P，Platform for Privacy Protection）工作组的隐私指导原则来建立可信计算平台的隐私模型。W3C 的 P3P 指导原则的主要内容包括以下几项。

（1）通知与通信：服务提供商应该及时、有效地把他们采集和使用用户信息的行为告知用户，用户代理应该为用户提供有效的工具，使用户能够获取服务提供商发布的通知，从而根据这些通知做出自己的决定。

（2）选择与控制：应该使用户具有足够强的能力，从而能够针对用户个人信息的采集、使用和透露做出明确的选择。

（3）公平性与完整性：用户应该拥有控制自己的个人信息的权力，能够确定在什么样的条件下共享相关的信息，服务提供商应该公平、完整地对待用户和处理用户的个人信息，以便能够保护隐私，促进信任。

（4）机密性：应该提供合理的安全保护机制，使用户的个人信息总是得到保护，从而维护用户个人信息的敏感性。

平台的身份信息属于可标识的个体信息（PII，Personally Identifiable Information），相当于用户的个人信息，TCG 采用 P3P 的以上指导原则保护平台的身份信息。

TCG 保护平台的身份，防止未知的实体或没有授权的实体轻易发现平台的身份信息。平台的背书密钥（EK）反映平台的真实身份，是一个绑定到平台上的非对称密钥对，用于认清一个平台，但不用于标识一个平台的身份。

TCG 保护平台的身份标识信息的方法是只有得到平台的属主或用户的许可才透露相应的信息。TCG 为平台身份引入了别名，就是平台的工作身份密钥（AIK），用于关联与特定的用途或特定的域相对应的信息。平台的属主或用户可以对 AIK 进行回收。

可以把 TPM 设为未启用或未激活模式，处于这样的模式时，TPM 不可能透露 EK 或 AIK。这些操作模式由平台的属主和操作员进行控制，因而，可以实现一种隐私策略，达到平台的对外证明能力与隐私保护之间的平衡。

另外，平台的属主可以把属主的授权数据与 EK 关联起来，这样，只有在提供了正确的授权数据的情况下，才允许透露 EK 的公钥。

3.6　可信计算平台的运行模型

TPM 的运行模型需要从 TPM 初次通电时开始考虑，这时，往往还处于 TPM 的生产过程中，TPM 还没有交到用户手上。TCG 为 TPM 定义了多种运行模式（也称为状态），以适应在多种应用场景中对 TPM 进行部署和维护。

以 TPM 的配置模式为基础，可以确定平台的配置模式，从而确定系统的引用监控机和应用程序的运行模型。

在考虑平台的运行模型问题时，也要考虑系统组件与 TPM 及其支持的服务之间的相互作用问题。

下面介绍 TPM 的工作状态、平台的工作方法、平台的软件接口和 TPM 命令的授权验证等方面的问题。

3.6.1　TPM的工作状态

以下相互独立的操作模式可以制约 TPM 的行为。

（1）启用/不启用：在系统的一个引导周期内，可以对 TPM 进行多次启用和不启用的设置。设置为不启用时，除了报告 TPM 的能力和接受对 PCR 寄存器的更新外，TPM 限制其他所有的操作；设置为启用时，TPM 的所有功能都起作用。

（2）激活/不激活：不激活与不启用相似，但它允许改变 TPM 的操作状态（即改变属主，或现场激活）。设置为激活时，TPM 的所有功能都起作用。

（3）已建立属主/未建立属主：TPM 中的 EK 是否已存在，标志着平台的属主是否已经建立，如果 TPM 中已经有 EK，则属主已经建立；如果 TPM 中还没有 EK，则属主尚未建立。平台的属主可以执行包括改变操作状态在内的所有操作。

这些模式是相互独立的，但可以有重叠的效果。如果同时设置了两种模式，其中一种模式使得 TPM 的某些功能起作用，另一种模式使得 TPM 的对应的功能不起作用，则优先权赋予使得功能不起作用的模式。

TPM 的很多命令对管理操作状态的策略有控制作用，这些命令由一些状态变量进行限定，这些状态变量限制对设置操作状态的命令的访问。例如，有些命令要求操作员必须在现场进行操作，借助现场操作来控制对平台的激活、启用和属主建立等状态的改动。

平台的制造商应该为平台配置默认的操作模式，该模式能够使系统处于一种保险而灵活的状态下，可方便地对属主进行配置，表 3-1 给出的就是这样的一种模式。

表 3-1　建议平台制造商设置的默认操作模式

操 作 状 态	默　认　值	可通过远程操作修改
可由属主启用	False	Yes
可由任意用户启用	False	No
可以建立属主	True	No
已经建立属主	False	No
已永久激活	False	No
已临时激活	False	No

表 3-1 描述了操作状态和需要现场改变操作模式的输入控制状态，从本质上说，除了由平台属主执行的状态改变之外，其他所有的状态改变都要求通过现场操作来实施。

可以把平台的 TPM 临时设置为不激活，这可以通过按下键盘上的一系列按键的方法来实现，系统复位时，可以把 TPM 重新设置为激活。临时激活不影响永久激活的值，它为操作员在短时间的交互作用后把 TPM 设置为不激活提供了方便。

下面通过几个例子说明在不同的平台部署场景中设置 TPM 的操作模式的方法，在这些例子中，假设所有的平台在出厂时都是按照如表 3-1 所示的默认操作模式进行配置的。

例 3.2　设一个可信计算平台归一个 IT 企业所有，由该 IT 企业负责管理，试说明使用该可信计算平台涉及的 TPM 操作模式配置情况。

解答：设该 IT 企业的资产管理部门接收该平台，将其登记造册，平台的销售商为 TPM 生成 EK，资产管理部门建立 TPM 的属主，同时确定属主的授权数据，授权数据和 EK 的公钥记录到资产登记册中，此时，属主已经建立，可关闭“可以建立属主”标记，TPM 的操作模式如下：

- 可由属主启用 = False；
- 可由任何人启用 = False；
- 可以建立属主 = False（已被改变）；
- 已经建立属主 = True（已被改变）；
- 已永久激活 = False；
- 已临时激活 = False。

把平台送到使用地点，连接到企业网络中，由资产管理部门执行远程启用，部门的技术人员对软件和管理功能进行配置，然后激活 TPM。此时，TPM 的操作模式如下：

- 可由属主启用 = True（已被改变）；
- 可由任何人启用 = False；
- 可以建立属主 = False；
- 已经建立属主 = True；
- 已永久激活 = True（已被改变）；
- 已临时激活 = False。

平台已经准备就绪，可以把它交给使用者使用了，此时，TPM 各种功能都能够正常作用，可以进行完整性度量，可以报告平台配置。

例 3.3　设一个可信计算平台归一个客户所有，试说明使用该可信计算平台涉及的 TPM 操作模式配置情况。

解答：销售商把平台卖给客户，并帮助客户建立属主，同时避免掌握属主的口令字符串。客户选择不启用 TPM。至此，属主已经建立，可关闭“可以建立属主”标记，其他操作模式没有变化，TPM 的操作模式如下：

- 可由属主启用 = False；
- 可由任何人启用 = False；
- 可以建立属主 = False（已被改变）；
- 已经建立属主 = True（已被改变）；
- 已永久激活 = False；
- 已临时激活 = False。

例 3.4　设一个可信计算平台归一个客户所有，通过外包服务的方式进行管理，试说明使用该可信计算平台所涉及的 TPM 操作模式配置情况。

解答：销售商把平台卖给客户，把平台的管理服务外包给专业的服务公司。服务公

司建立属主，并把平台设置为可进行远程管理的状态。由客户决定是否激活 TPM。此时，TPM 的操作模式如下：

- 可由属主启用 = True（已被改变）；
- 可由任何人启用 = False；
- 可以建立属主 = False；
- 已经建立属主 = True（已被改变）；
- 已永久激活 = False；
- 已临时激活 = False。

客户只有在与外包服务公司沟通时才能使用 TPM 的功能，使平台复位，并在可以执行管理功能时激活 TPM，若想以匿名方式使用 Internet，则可按下键盘上的一系列按键，把 TPM 设置为临时不激活。此时，TPM 的操作模式如下：

- 可由属主启用 = True；
- 可由任何人启用 = False；
- 可以建立属主 = False；
- 已经建立属主 = True；
- 已永久激活 = True（已被改变）；
- 已临时激活 = True（已被改变）。

当客户使用 Internet 时，管理行为暂被阻止，客户重新启动计算机时，TPM 回到已永久激活状态，管理行为又恢复正常。

当平台管理的外包服务合同到期时，客户可以换服务公司，在这种情况下，需要把“可以建立属主”的标记设为 True，然后，让新的服务公司执行建立属主操作，成为新的属主。

以上几个例子仅展示了平台的几种配置选项，不同的平台部署模型具有不同的最佳操作模式配置方法。

3.6.2　平台的工作方法

当平台开始操作时，它必须从 RTM 开始启动，RTM 掌握它自己及其他可信构件块的信息，这里所说的可信构件块就是 RTS 和 RTR 等。TPM 的启动状态等价于系统的初始化状态，当平台接通电源开始启动时，RTM 必须给 TPM 发信号，指示它启动初始化过程。TPM 的初始化过程包括 TPM 的自检过程，该自检过程检测该 TPM 的各项功能是否正常。

不同的电源管理模式可以影响 TPM 的初始化过程，RTM 负责选择和控制最合适的 TPM 初始化过程，TPM 会根据 RTM 的操作做出相应的响应。以下是 TPM 设备可能遇到的电源管理模式。

- 初始通电：TPM 必须把所有的 PCR 寄存器设置为零，并使控制标记复位；
- 睡眠：TPM 必须把易失性状态保存到永久存储器中；
- 恢复：TPM 必须恢复此前保存的状态；
- 休眠：TPM 没有与此相对应的电源状态，系统必须保存易失性状态。

以上这些操作是与平台密切相关的，不同的平台可能实施不同的具体操作方法，平台相关的规范必须详细描述各种有效的平台状态及相应的 TPM 响应方法。

作为系统初始化操作的一部分，系统将对平台的组件和配置进行度量，不过，初始阶段的度量操作既不检测是否存在不保险的配置也不采取任何行动去阻止初始化过程的继续推进，这样的任务应该由系统中的引用监控机（如操作系统等）负责完成，

初始阶段的度量完成之后，下一步的度量将由操作系统的程序装载器实施，它在把一个程序装载到内存之前对该程序进行度量。由于操作系统帮助实施系统的完整性，对于描述软件配置中不良状态的策略，可以由操作系统的程序装载器进行度量和实施。

平台可以对应用程序进行度量，应用程序也可以对平台进行检查。应用程序可以拥有描述可信平台配置的策略，因而，它可以辨别平台的配置是否可信，如果发现有不可信的情况，它可以拒绝与平台进行交互，或者限制与平台之间的交互。

这种平台与应用程序之间的双向度量的语义可以扩展到其他平台，进而为分布式应用程序的度量提供支持。分布式应用程序的各协作方在进行协作之前可以进行跨平台的交叉检查，从而确保分布式协作的顺利进行。

3.6.3　平台的软件接口

TCG 采用通用计算平台中常用的软件服务的结构层次划分思想，为可信计算平台定义了三层软件接口，它们分别是设备驱动程序库接口（TDDLI，TPM Device Driver Library Interface）、软件核心服务接口（TCSI，TSS Core Service Interface）和服务提供方接口（TSPI，TCG Service Provider Interface），可信计算平台软件层次结构如图 3.8 所示。

在图 3.8 给出的可信计算平台软件的层次结构中，从低层到高层，TCG 定义了六层内容，底层是 TPM，顶层是应用程序，由低到高的其他各层依次是 TPM 设备驱动程序、设备驱动程序库、软件核心服务和服务提供方。其中，TPM 及其设备驱动程序是内核模式的内容，其他都是用户模式的内容。

在三层接口中，最下层的接口是 TDDLI，这是上层软件访问设备驱动程序库的接口，属于用户模式的接口，与直接访问内核模式的设备驱动程序的接口相比，这样的接口有以下优点：

（1）它使得 TCG 软件栈（TSS，TCG Software Stack）的任何一种实现都能够与任何一种 TPM 进行正确的通信；

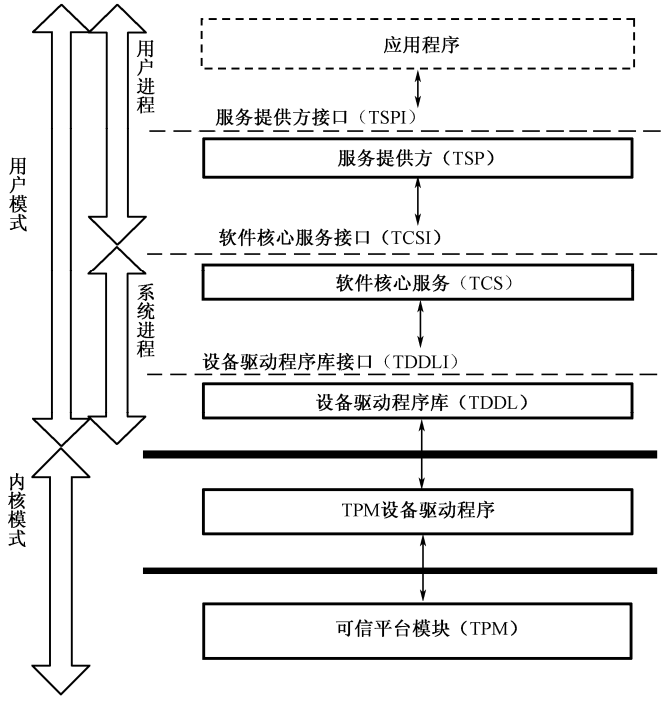


图 3.8　可信计算平台软件层次结构

（2）它能够为 TPM 应用程序提供一种独立于操作系统的接口；

（3）它使得 TPM 的销售商可以向用户提供用户模式的 TPM 软件仿真程序。

设备驱动程序库（TDDL）实现用户模式与内核模式之间的切换。不过，TDDL 不处理线程级别的软件与 TPM 的交互，也不对 TPM 命令进行系列化处理，这些任务由软件栈中的高层组件去完成。另外，TPM 不支持多线程的访问，只接受单线程的访问，所以，对于每个平台，在任何确定的时刻，只能有一个 TDDL 实例存在。

中间层的接口是 TCSI，它是上层软件访问一组通用的平台服务的接口。虽然在一个平台上可以存在多个服务提供方，但是，TCSI 为它们提供共同的软件核心服务，使它们可以呈现共同的行为。软件核心服务层提供以下几个核心服务。

- 环境管理：实现线程级的 TPM 访问，支持线程访问 TPM；
- 凭证与密钥管理：存储与平台关联的凭证和密钥；
- 度量事件管理：管理事件日志记录，管理对关联的 PCR 寄存器的访问；
- 参数块生成：负责 TPM 命令的系列化、同步和处理。

软件核心服务层以系统进程的形式在用户模式中工作，可以相信它能够管理用户向 TPM 提供的授权信息。

最上层的接口是 TSPI，这是应用程序访问 TPM 的 C 语言接口。服务提供方（TSP）层与应用程序在相同的进程地址空间中工作，有关用户授权方面的工作在这一层进行。授权操作可以通过本层的代码提供的用户接口实现，也可以通过软件核心服务层的回

调（Callback）机制实现（如果是远程调用的话）。为了给最终用户提供一个一致的授权接口，本地应用程序不必提供授权服务，它们可以依靠平台中固有的服务来完成授权操作。

TSP 层提供环境管理和加密处理两种服务。环境管理器通过动态的句柄来为应用程序和 TSP 资源的有效利用提供支持，每个句柄为一组相关的 TCG 操作提供一个环境，应用程序中的不同线程可以共享同一个环境，每个线程也可以拥有自己的独立环境。

为了全面利用 TPM 提供的保护功能，系统应该提供加密支持功能，不过，TSP 层只提供平台操作所必需的相应支持，典型的支持功能包括消息摘要计算和字节流生成等。TSP 层没有提供诸如大宗数据加密等方面的接口。

设备驱动程序接口（TDDLI）、软件核心服务接口（TCSI）和服务提供方接口（TSPI）确定了 TCG 的可信计算平台的软件接口规范。根据这些接口规范，可以方便地建立可信计算平台的软件系统及基于可信计算平台的应用系统。

TCG 的可信计算平台可以在新的应用基础架构上为应用系统提供支持，也可以在现有应用基础架构上为应用系统提供支持。图 3.9 给出了基于 TCG 的可信计算平台的三种典型的应用支持方案。

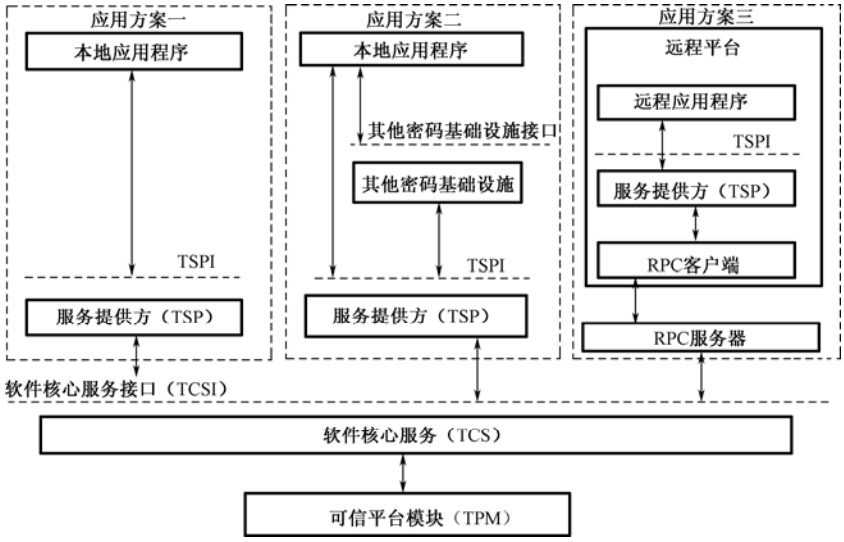


图 3.9　可信计算平台典型应用支持方案

在应用支持方案一中，应用系统直接建立在 TCG 的可信计算平台之上。在方案二中，应用系统建立在 TCG 的可信计算平台与一个现有的加密与安全服务的基础架构之上。在方案三中，远程应用系统建立在 TCG 的可信计算平台与远程过程调用（RPC，Remote Procedure Call）机制之上。

下面借助例子来简要说明应用系统通过各种接口获取可信计算平台提供的服务的基本方法。

例 3.5　设一个应用系统直接建立在 TCG 的可信计算平台之上，应用程序根据 TPM 中的 PCR 寄存器的值和验证凭证中的值验证平台的配置，试说明验证的过程。

解答：验证过程包含以下步骤：

- 对应用程序对象进行初始化，为读取 PCR 寄存器的值做准备；
- 读取寄存器 PCR5 的值；
- 对比 PCR 寄存器中的值与验证凭证中的值。

在应用程序层、服务提供方层、软件核心服务层和 TPM 层需要实施的操作如下：

（1）应用程序层。

```
{
Tspi_Contest_Create( &hContext);
Tspi_Context_Connect(hContext, NULL);
Tspi_Context_GetTpmObject(hContext,&hTPM);
Tspi_TPM_PcrRead(hTPM, 5, &ulPcrValueLength, &rgbPCRValue);
Compare(correctValueOfPCR5, rgbPCRValue);
}
```

（2）服务提供方层。

当 Tspi_TPM_Quote()被调用时，软件核心服务层要求把用于对 PCR 寄存器值签名的密钥装载到 TPM 中，这可以由 Tspi_Context_LoadKeyByUUID 命令完成。

还需要生成一个能够表示要读取的 PCR 寄存器值的 PcrComposite 索引，这可由 Tspi_PcrComposite_SelectPcrIndex 命令完成。

该命令执行时，要调用 Tcsip_Quote 命令。

Tcsip_Quote 命令实现对 PCR 寄存器值的签名和检索。

```
{ …
hKey = Tcsip_LoadKeyByBlob(ikey);          //装载由哈希值标识的密钥
Tcsip_Quote(hKey, …);                      //检索签过名的 PCR 寄存器值
… }
```

软件核心服务层的命令将通过设备驱动程序库命令调用设备驱动程序。

（3）软件核心服务层。

```
{ …
loadKeyMsg = PBG_LoadKey(hKey);            //编制 TPM_LoadKey 命令
quoteMsg = PBG_Quote();                    //编制 TPM_Quote 命令
Tddli_Open();                              //打开 TPM 通信信道
Tddli_TransmitData(loadKeyMsg);            //发送命令并接收响应
Tddli_TransmitData(quoteMsg);              //发送命令并接收响应
Tddli_Close();                             //关闭与 TPM 通信的信道
… }
```

（4）TPM 层。

接收到装载密钥（loadKeyMsg）和检索 PCR 寄存器值（quoteMsg）的消息时，TPM 顺序地对命令块进行分析，并执行相应的操作。

例 3.6　设一个应用系统建立在 TCG 的可信计算平台与一个现有的加密与安全服务的基础架构之上，试说明可信计算平台对应用系统的支持方法。

解答：可以调整现有的加密与安全服务基础架构（如 PKCS11 加密令牌接口标准的架构）中的固定令牌或智能卡存储功能接口，以便使现有架构能访问 TPM 设备，并利用 TPM 来存储和检索现有架构中需要使用的密钥（对称密钥或非对称密钥）。

例 3.7　设一个远程应用系统通过本地的 TCG 可信计算平台获得可信计算支持，试说明远程应用系统读取本地平台上的 PCR 寄存器值的方法。

解答：远程的应用程序与本地的 TPM 进行交互。设应用程序直接与软件核心服务层（TCS）对话，TCS 通过远程过程调用（RPC）机制实现。远程应用程序向它的宿主系统的 TCS 发出读取 PCR 寄存器值的命令，该 TCS 通过它的参数块生成器编制 TPM 消息（loadKeyMsg 和 quoteMsg），并把它们传给 RPC 机制。

远程系统的 TCS 执行的操作如下。

```
{ …
loadKeyMsg = PBG_LoadKey(hKey);          //编制 TPM_LoadKey 命令
quoteMsg = PBG_Quote();                  //编制 TPM_Quote 命令
RPC_Open();                              //打开 TPM 通信信道
RPC_Send(loadKeyMsg);                    //发送命令并接收响应
RPC_Send(quoteMsg);                      //发送命令并接收响应
RPC_Recv(loadKeyMsg);                    //发送命令并接收响应
RPC_Recv(quoteMsg);                      //发送命令并接收响应
RPC_Close();                             //关闭与 TPM 通信的信道
… }
```

远程系统的 RPC 机制把命令发送给本地系统的 RPC 机制。本地系统的 RPC 机制接收到远程系统发来的信息后，把它们传递给本地系统的 TCS 进行处理。

本地系统的 TCS 执行的操作如下。

```
{ …
RPC_Recv(loadKeyMsg);                    //接收命令
RPC_Recv(quoteMsg);                      //接收命令
Tddli_Open();                            //打开 TPM 通信信道
Tddli_TransmitData(loadKeyMsg);          //发送命令并接收响应
Tddli_TransmitData(quoteMsg);            //发送命令并接收响应
Tddli_Close();                           //关闭与 TPM 通信的信道
```

```
RPC_Send(loadKeyMsg);                    //发送应答信息
RPC_Send(quoteMsg);                      //发送应答信息
… }
```

这样，远程系统的 TCS 把请求信息发送给本地系统的 TCS，本地系统的 TCS 把请求信息传递给本地系统的 TPM，本地系统的 TPM 执行相应的命令，并通过本地系统的 TCS 和远程系统的 TCS 把执行命令后产生的应答信息发送给远程应用程序。

以上三个例子展示了可信计算平台的软件接口在实际应用中的基本用法，同时展示了 TPM 功能的基本用法。

可信计算平台中的 TPM 功能可以由本地应用程序直接使用，也可以由本地应用程序通过其他组件间接使用，还可以由远程应用程序远程使用。

3.6.4　TPM命令的授权验证

在可信计算平台中，TPM 外部的实体通过给 TPM 发出命令而获得 TPM 提供的功能。实体向 TPM 发出的所有对安全或隐私有影响的命令或透露平台秘密的命令必须在得到授权的情况下才能执行。

定义 3.9　确定一个发出 TPM 命令的实体是否拥有执行该命令的合法授权的过程称为 TPM 命令的授权验证。

为了通过命令的授权验证，实体在发出命令的同时必须提供一个与该命令相关的秘密，是否知道该秘密是衡量实体是否拥有相应授权的准则。

1．命令验证方法

任何实体（如进程、线程或嵌入式控制器等）都可以向 TPM 发出命令，为了使命令能够在可信的环境下产生可信的效果，平台需要在实体与 TPM 之间建立一个安全的通信信道，用于实施命令的发送和结果的返回。

可以采用面向会话的消息交换方法中的“请求–响应”语义来实现实体与 TPM 之间的安全通信信道，通过建立会话来确保实体在合法授权的约束下访问 TPM 的对象（如密钥、密钥块和离散部件等），一个会话对应一个安全的通信信道。命令授权验证会话如图 3.10 所示，说明了在实体与 TPM 之间建立用于进行命令授权验证的会话的基本原理。

由图 3.10 可知，一个会话由一个会话标识、一个即时随机数、一个消息摘要和一个瞬时秘密构成。一个会话的两个端点拥有相同的会话标识、不同的即时随机数。消息摘要与在两个端点之间交换的消息相对应。瞬时秘密用于把消息绑定到 TPM 中一个明确的命名对象上，也可以用于在消息交换过程中实现加密。

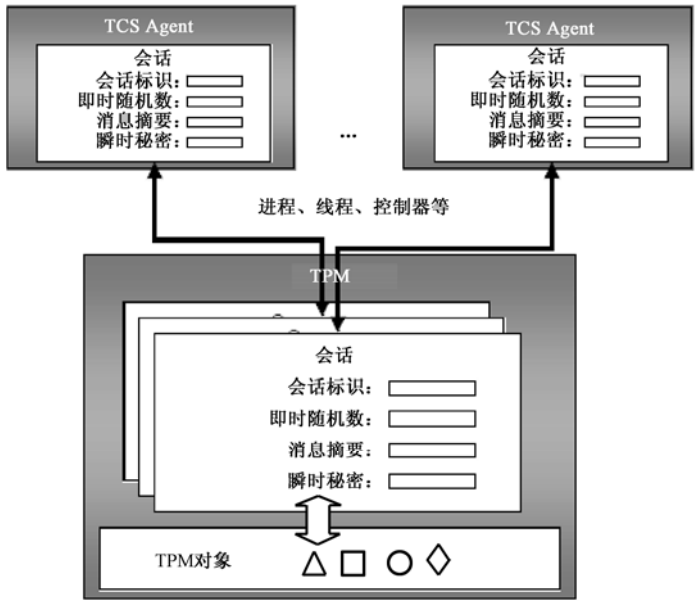


图 3.10　命令授权验证会话

为了成为会话的一个端点，从而加入授权的消息交换的行列，实体必须提供一个同时用于进行身份认证和授权验证的秘密，该秘密通常是一个 160 比特的口令字符串，也称为授权数据。选择长度为 160 比特是为了与 SHA-1 运算结果的长度保持一致。

授权数据可以与 TPM 的对象、TPM 本身或命令接口相关联，分别用于验证访问 TPM 的对象、访问 TPM 或执行相应命令的合法性。

为了实现执行 TPM 命令时的授权验证，有必要定义有效的命令验证协议。命令验证协议可以使用基于哈希的消息认证码（HMAC，Hashed Message Authentication Code）来建立实体与 TPM 间的授权会话。

在会话双方共享密钥的前提下，HMAC 可用于确定在会话中交换的消息是否已被篡改。消息的发送方计算原始数据的哈希值，并将原始数据和 HMAC 放在一个消息中传送。消息的接收方接收到消息后，重新计算所接收的数据的哈希值，检查计算所得的哈希值是否与传送的哈希值匹配。

在授权的会话中交换的一个消息可以表示成包含以下三部分内容的一个框架。

- 消息容器：标识消息的类型、长度和格式；
- TPM 命令：命令名、输入/输出参数和返回代码；
- 会话状态：会话标识、控制标志和会话消息的摘要值。

在“请求–响应”协议中，会话中的 TPM 和发出命令的实体在把会话向前推进一步之前，都会对会话消息进行验证。会话状态中包含即时随机数，可以避免会话中的重放攻击。

2．命令验证协议

TCG 为实现 TPM 命令的授权验证定义了五个协议，分别称为授权数据插入协议（ADIP，Authorization Data Insertion Protocol）、授权数据修改协议（ADCP，Authorization Data Change Protocol）、非对称授权修改协议（AACP，Asymmetric Authorization Change Protocol）、对象无关的授权协议（OIAP，Object-Independent Authorization Protocol）和对象相关的授权协议（OSAP，Object-Specific Authorization Protocol）。

其中，ADIP、ADCP 和 AACP 用于创建和管理授权信息，这些信息包含在 TPM 控制下的对象中；OIAP 和 OSAP 用于建立由前面三个协议支撑的授权的会话环境。

ADIP、ADCP 和 AACP 的语义紧密地集成在 TPM 的管理命令之中，TPM 没有为其提供一般化的接口。

OIAP 和 OSAP 的使用方法与 TPM 的一般命令类似，由 TPM 的命令解释器负责解释，它们拥有对应的 TPM 命令接口和系列化方法。TPM_OIAP()和 TPM_OSAP()命令用于对会话对象进行初始化，初始化可以得到会话对象的相应句柄，需要进行授权验证的 TPM 命令将会引用相关的句柄。

OIAP 可以在 TPM 与一个外部实体之间建立一个授权的明文会话，软件核心服务层（TCS）为 OIAP 会话的管理提供了有力的支持。OIAP 的消息交换过程如图 3.11 所示。

图 3.11 描述了 TCS、OIAP 会话和 TPM 三者之间的协同作用。如果把 TPM 所在的平台看做本地平台，则 TCS 和 OIAP 对象既可以在本地平台上，也可以在远程平台上。TCG 重点关注 TPM 端点的保护，同时兼顾 TCS 端点的安全。OIAP 消息交换主要发生在 OIAP 会话与 TPM 之间，TCS 消息交换提供了一个环境，该环境描述如何使用 OIAP 会话。

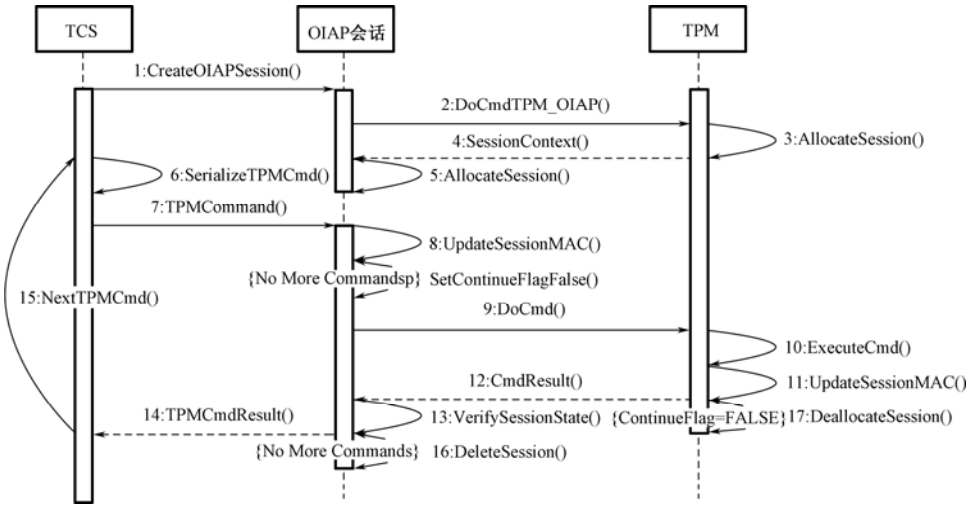


图 3.11　OIAP 的消息交换过程

图 3.11 中的第 1～5 步建立一个 OIAP 会话，第 16 步和第 17 步清除一个会话，在一个会话中，可以处理多个 TPM 命令，如第 6～15 步所示。TCS 代理可以确定什么时候将不再有新的命令发出，于是在执行最后一个命令的消息中设置一个标志，以触发会话的清除操作。

OIAP 通过为所有交换的消息计算消息认证码（MAC）来确保消息的完整性，它使用 HMAC 方法，以一个秘密和一个消息数据的哈希值为输入，产生一个 20 字节的输出。

输入中的秘密就是即时随机数，OIAP 定义了两种即时随机数，分别用于一个会话的两个端点，图 3.11 中的会话端点在第 4 步和第 9 步交换即时随机数值，第 4 步还提供会话标识。

输入中的秘密与消息数据的组合的哈希值构成了会话 MAC。一个会话包含如下两种逻辑结构。

（1）TPM 命令：包括参数和返回码，MAC 计算中实际使用的是 TPM 命令域的 SHA-1 哈希值；

（2）会话配置参数：包括会话标识、即时随机数和控制标志，在 HMAC 计算中会用到会话配置参数。

OSAP 的思想与设计和 OIAP 非常相似，它在 OIAP 语义的基础上增加了新的约束，一方面把授权的会话绑定到 TPM 的对象上，另一方面使用了瞬时秘密。OSAP 的消息交换过程与 OIAP 的消息交换过程的区别在于 OSAP 对 OIAP 消息交换过程中的以下两个步骤进行了扩充：

（1）在第 2 步中标识 TPM 的目标对象，并提供第三个即时随机数（在 OIAP 使用了两个即时随机数的基础上）；

（2）在第 4 步中提供第四个即时随机数。

增加的即时随机数用于为会话生成瞬时秘密，该瞬时秘密用于计算 MAC，还可以用于对数据进行加密。

OSAP 对于更新与 TPM 所管理的对象相关联的敏感信息特别有用。

ADIP 用于为 TPM 管理的对象建立新的实例，TPM_CreateWrapKey()命令使用该协议。创建新的对象实例时，发出命令的实体可以提供一个口令字符串，由 ADIP 把该口令字符串与新建立的对象实例关联起来，这样，要访问该对象实例，必须提供相应的口令字符串，从而实现对象实例的授权访问。

ADIP 为 OSAP 提供支撑，以便在实体与新对象实例（即子对象实例）的父对象实例之间建立授权的会话，使用 ADIP 创建对象如图 3.12 所示。

子对象实例的授权数据通过 XOR 运算借助 OSAP 的共享秘密进行加密，TPM 借助该共享秘密通过 XOR 运算对授权数据密文进行解密。

ADCP 用于更新 TPM 管理的对象的授权数据，TPM_ChangeAuth()命令使用该协议，该协议为 ADIP 提供隐私性和完整性支撑，并为对子对象实例的授权访问增加

OIAP（或 OSAP）会话。使用 ADCP 更新子对象的授权数据如图 3.13 所示。

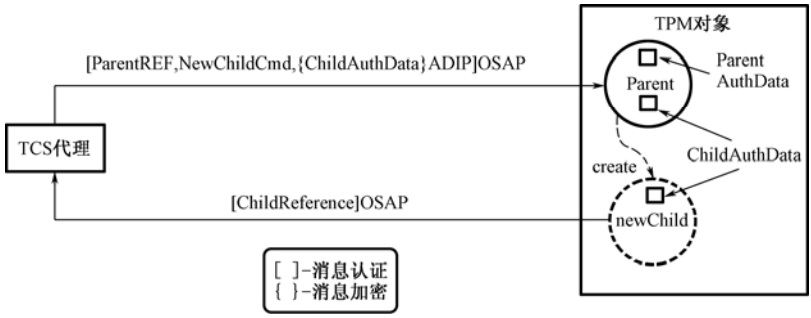


图 3.12　使用 ADIP 创建对象

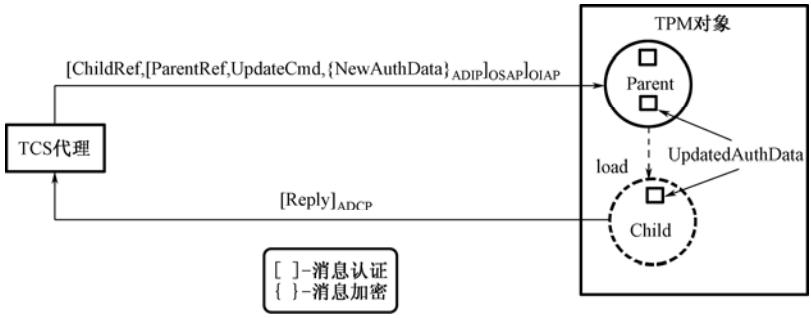


图 3.13　使用 ADCP 更新子对象的授权数据

TPM 管理的大多数对象都能适应“父-子”关系模型，但也有例外。例如，从逻辑意义上说，SRK 是存储子系统的根，对于此类情形，TPM 可以利用 EK 授权数据为 SRK 建立 ADIP 会话，进而兼容 ADCP 协议。

AACP 用于实现在不让父对象实例知道子对象实例的授权数据的情况下更新子对象实例的授权数据，TPM_ChangeAuthAsymStart()和 TPM_ChangeAuthAsymFinsih()命令使用该协议。

当多个实体共享父对象实例的授权数据时，如果某些实体没有获得访问子对象实例的授权，那么，使用 ADCP 就存在一定的安全威胁，对 SRK 的直接子对象实例的访问很有可能出现这种情形，AACP 可以防止这样的威胁。

3.7　可信计算平台的编程接口

第 3.6.3 小节介绍了 TCG 定义的可信计算平台的软件接口，这为理解可信计算平台的编程接口建立了很好的基础。本节重点介绍 TPM 的编程接口。首先，有必要简要浏览一下 TCG 规范中与程序设计有关的命名习惯。

3.7.1　编程相关的TCG命名习惯

在 TCG 规范中，命令（Command）、函数（Function）、操作（Operation）和接口（Interface）是几个常用的术语，它们的含义如下。

- 命令：命令是 TPM 对外提供的功能，可以由 TPM 命令处理器解释，每个命令与一个唯一的命令序号相对应；
- 函数：函数是 TPM 之外的其他模块提供的功能，具有程序设计接口；
- 操作：操作是一系列步骤或协议流程的集合，可以由一个或多个命令或函数实现；
- 接口：接口是通往某个特定模块的命令或函数的入口点，包括参数和返回码。
- 根据安全的相关性，TCG 又把操作划分为以下几种类型。
- 保护操作：指对平台的安全特性有影响的操作，对应所有的 TPM 命令，TPM 命令的接口名以“TPM_”开头；
- 非保护操作：指支持 TPM 的保护功能的操作，但不是 TPM 命令，对应在 TPM 的外部实现的函数，函数接口名以“Tspi_”、“Tsci_”或“Tddli_”开头，分别与服务提供方接口、软件核心服务接口和设备驱动程序库接口相对应；
- 连接操作：指实现平台与 TPM 绑定的操作，实现连接操作的命令由平台相关的规范定义，这些命令的接口名以“TSC_”开头。例如，TSC_PhysicalPresence() 命令，其中，TSC 表示 TCG 软件连接（TCG Software Connection）。

3.7.2　程序员视角的TPM结构

为了能够在程序设计中有效地应用 TPM 提供的功能，有必要以程序员的眼光去了解 TPM 的基本情况。从程序员的视角看到的 TPM 的结构可以表示为如图 3.14 所示的形式。

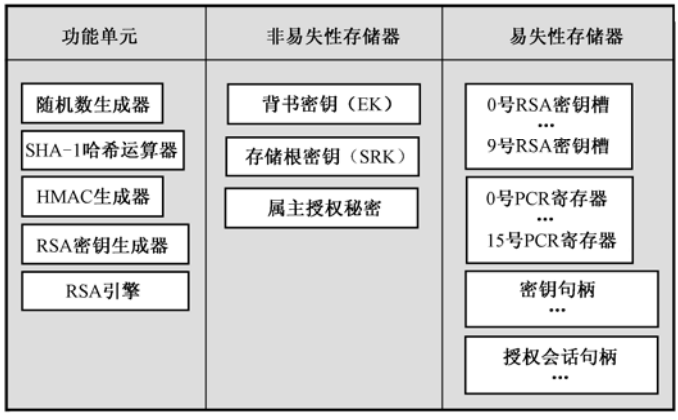


图 3.14　程序员视角的 TPM 结构

根据图 3.14 的表示，TPM 包含五个加密功能单元，分别是随机数生成器、SHA-1 哈希运算器、HMAC 生成器、RSA 密钥生成器和 RSA 引擎。随机数生成器为片上密钥生成提供高质量的随机数源，也可供应用程序使用。SHA-1 哈希运算器提供完整性度量所需的哈希值。HMAC 生成器用于计算消息认证码。RSA 密钥生成器以随机数生成器提供的随机数为基础，生成 2048 比特的 RSA 密钥。RSA 引擎可以执行签名、加密和解密操作。

TPM 把三个重要密钥存放在非易失性存储器中。背书密钥（EK）是 2048 比特的 RSA 公钥和私钥对，出厂前在片上随机生成，以后不可改变。EK 的私钥永远不会离开 TPM 芯片，EK 的公钥可用于实施对外证明，在给 TPM 发送敏感数据时，还可以用它对敏感数据进行加密。例如，执行 TPM_TakeOwnership()命令时，可以用 EK 的公钥对授权数据进行加密。从隐私的角度看，EK 是敏感的密钥，所以，TPM 的属主可以把它设置为彻底禁用。

存储根密钥（SRK）是 2048 比特的 RSA 密钥对，初始的 SRK 是空的，它由 TPM_TakeOwnership()命令创建。该密钥永远不离开 TPM 芯片，它的作用是对 TPM 生成的私钥进行加密，以便能把它们保存到 TPM 外部的存储器中，存储在 TPM 外部的私钥装载到 TPM 中时，它又对它们实施解密操作。TPM 的属主可以清除 SRK。

属主授权秘密是一个 160 比特的秘密，由属主和 TPM 共享，执行 TPM_TakeOwnership()命令时，属主把它装载到 TPM 中，它用于验证敏感的属主命令请求。

易失性存储器为 RSA 密钥对提供临时存储槽，加密后存储在 TPM 外部的密钥可以装载到这些存储槽中，以供使用。虽然装载到存储槽中的密钥是易失性的，关电后不复存在，但有时其中可能是存有密钥的，因此，在装载新的密钥时，需要清除槽中的旧密钥，以为新密钥腾出空间。

易失性存储器也包括平台配置寄存器（PCR），这些寄存器用于保存 160 比特的软件完整性度量值（即哈希值）。系统引导期间，BIOS、扩展 BIOS、主引导记录（MBR）、操作系统装载程序（如 GRUB）和其他指定的文件（如操作系统内核）等的完整性度量值将被生成，这些度量值将被保存到各种 PCR 寄存器中。通电引导时，BIOS 主动使所有 PCR 寄存器的值复位。当系统从一个挂起的状态恢复时，BIOS 启动 TPM，并使它恢复已保存的 PCR 寄存器的值，为了能够实施恢复操作，在进入挂起状态前，TPM 的设备驱动程序必须执行 TPM_SaveState()命令。

易失性存储器还用于存放两类句柄：密钥句柄和授权会话句柄。密钥句柄为装载到 TPM 中的密钥提供临时的名称，以便后面的命令能够指定应该使用哪个密钥。如果装载了多个密钥，则在清除密钥时应该清除相应的句柄。授权会话句柄用于在多个命令中标识授权状态数据，它们由 TPM_OSAP 和 TPM_OIAP 授权命令创建。当不再需要这些授权会话句柄时，应该用 TPM_Terminate_Handle()命令将它们清除。

3.7.3　TPM的启动与清零

在考察具体的 TPM 命令之前，我们应该探究一下如何让 TPM 开始工作。情况其实很简单，BIOS 可以负责 TPM 的启动和清零工作。通电时，TPM 即被激活，但没有被启动。此时，BIOS 必须向它发出一个 TPM_Startup()命令，该命令可以完成以下工作中的一项：

- 将 TPM 置于不激活状态；
- 启动 TPM 并使 PCR 寄存器复位；
- 启动 TPM 并恢复曾经保存的 PCR 寄存器的值。

如果 BIOS 将 TPM 置于不激活状态，它将一直处于该状态之中，直到下一次重新通电，没有软件命令能够把它重新激活。系统引导时，可以以清除 PCR 寄存器值的方式启动 TPM，所以，在引导期间，可以正确地计算所有 PCR 寄存器的值。在挂起时刻，TPM 的设备驱动程序负责发出执行 TPM_SaveState()命令的请求，以确保在恢复时刻可以获得有效的 PCR 寄存器的值。

必要时，BIOS 也负责执行 TPM_ForceClear()命令，该命令使 TPM 完全复位，它卸载所有的密钥和句柄，清除 SRK 和属主授权秘密。该命令要求提供现场操作证明，在 IBM 个人计算机中，在系统上电时按住 Fn 键（键盘右下角的蓝色键）可以做到这一点。

仍以 IBM 个人计算机为例，可以在 BIOS 的配置模式中设置 BIOS 对 TPM 进行的激活与清零控制。为了能够在开机时启动 TPM，首先，按住 Fn 键；然后，打开计算机的电源开关，当 BIOS 屏幕出现时，释放 Fn 键，并按下 F1 键，进入 BIOS 配置模式，在配置菜单中进行以下选择：

Config → Security System → Enable and Clear

这些步骤启动 TPM 的操作，并把它清零，至此，已经为建立 TPM 的属主做好了准备。

3.7.4　在程序中使用TPM命令

TPM 命令接口采用与远程过程调用（RPC）的思想相似的“请求−响应”交互模型，一个命令解释器和一个参数块生成器负责对命令的请求和响应进行编码和解码。软件核心服务层（TCS）为 TPM 外部的实体提供这些类型的服务，TPM 的输入/输出组件为 TPM 提供相应的服务。

TPM 命令以命令消息的形式在实体与 TPM 之间传递。每个命令消息拥有一个消息头，它标识消息的类型和消息的长度（以字节为单位），消息的类型可以是“请求”或“响应”。

消息头的结构字段及占用的相应空间大小如下。

- 消息类型：2 字节。

- 消息长度（含消息头）：4 字节。

命令的语法以按次序排列的方式表示。每个命令对应一个唯一的序号，它明确地标识相应命令对应的 TPM 入口点和接口语义。命令序号后面是参数块（对请求消息而言）或返回码（对响应消息而言）。

命令请求消息的结构字段及占用的相应空间大小如下。

- 命令序号：4 字节。
- 参数块（参数 1，…，参数 *n*）：由命令序号的语义确定。

命令响应消息的结构字段及占用的相应空间大小如下。

- 返回码：4 字节。
- 参数块（参数 1，…，参数 *n*）：由命令序号的语义确定。

对于命令响应消息而言，参数块字段中存放的实际上是返回数据。

在 TPM 中，命令的请求与响应是作为一个原子操作进行处理的，即，当 TPM 接受了一个命令请求之后，在处理完相应的命令响应之前，它绝不会接受下一个命令请求。

如果一个消息属于经过授权的消息（如 OIAP 或 OSAP 会话消息），则该消息还拥有一个消息尾，消息尾包含 MAC 摘要、会话标识和即时随机数。

OSAP 会话请求的消息尾的结构字段及占用的相应空间大小如下。

- 会话句柄：4 字节。
- 即时随机数（请求方的）：20 字节。
- 标记：1 字节。
- 会话 MAC（请求方的）：20 字节。

OSAP 会话响应的消息尾的结构字段及占用的相应空间大小如下。

- 即时随机数（TPM 的）：20 字节。
- 标记：1 字节。
- 会话 MAC（TPM 的）：20 字节。

所有命令消息中的二进制值均以网络字节次序进行排列（低字节在前、高字节在后），字符型数据均以 ASCII 码形式表示。

根据上面介绍的命令消息结构（简单起见，暂不考虑拥有消息尾的情况），在程序设计中，可以把命令请求消息组织成命令包，把命令响应消息组织成响应包。

命令包的数据结构的一般形式可以定义如下。

- 16 比特：包的类型（即消息类型）。
- 32 比特：包的总长度（即消息长度）。
- 32 比特：TPM 命令序号。
- 可变长度：命令数据（即参数块）。

响应包的数据结构的一般形式可以定义如下。

- 16 比特：包的类型（即消息类型）。

- 32 比特：包的总长度（即消息长度）。
- 32 比特：返回码。
- 可变长度：返回数据（即参数块）。

向 TPM 发命令，就是把命令包传给 TPM；从 TPM 获取响应结果，就是从 TPM 接收响应包。命令包和响应包中的 16 比特和 32 比特的值是按网络字节次序排列的，程序在处理过程中，需要在网络字节次序与主机字节次序之间进行必要的转换。

要向 TPM 发出命令，首先必须把命令包构造出来，命令包构造完成后，命令包的长度便可以确定，因此，向 TPM 发出命令时，可以向 TPM 传送确定长度的命令包。

但是，从 TPM 获取响应结果时，只有在接收完响应包之后，才有可能知道响应包的实际长度，那么，在程序中，如何确定要从 TPM 中接收长度为多少的响应包呢？根据定义，我们可以知道 TPM 能处理的包的最大长度（如 4096 字节），因此，在程序语句中，可以试图从 TPM 中接收最大长度（如 4096 字节）的响应包，等接收响应包的操作完成后，再确定包的实际长度。

在 Linux 操作系统中，可以把 TPM 的设备驱动程序设计为一个可装载内核模块（如 tpm.o），它为 TPM 芯片提供一个字符设备接口，相应的设备文件可以是/dev/tpm。这样，向 TPM 发命令等价于向设备文件/dev/tpm 写（write）一个命令包，从 TPM 获取响应结果等价于从该设备文件读（read）一个响应包。

例 3.8　设在 Linux 操作系统中，TPM 对应的设备文件是/dev/tpm，试给出一个 C 语言函数，使它能够用于向 TPM 发送命令包并从 TPM 接收响应包。

解答：可用于发送命令包和接收响应包的函数 TPM_Transmit 定义如下（行号是为了说明方便而加的）：

```
1.  uint32_t TPM_Transmit(unsigned char *blob)
2.  {
3.      int tpmfp, len;
4.      uint32_t size;
5.      tpmfp = open("/dev/tpm", O_RDWR);
6.      size = ntohl(*(uint32_t *)&blob[2]);
7.      len = write(tpmfp, blob, size);
8.      len = read(tpmfp, blob, 4096);
9.      return(ntohl(*(uint32_t *)&blob[6]));
10. }
```

第 1 行定义的参数 blob 指向命令包的数据结构，第 5 行打开 TPM 对应的设备文件，第 7 行按照命令包的实际长度把命令包写入 TPM 中，第 8 行欲从 TPM 中读取 4096 字节长的响应包，读操作完成后，响应包的实际长度等于 len 的值。

通过 BIOS 的配置，使 TPM 启动并清零之后，如果操作系统中的 TPM 设备驱动程

序已经正常工作，就可以向 TPM 发出命令以获取可信计算支持了。作为起步，可以从简单的 TPM 命令开始，体验在程序中使用 TPM 功能的基本方法。

TPM_Reset()是 TPM 中最简单的命令，它可以清除存放在 TPM 中的所有授权会话的句柄，例 3.9 说明了它的用法。

例 3.9　设在 Linux 操作系统中，TPM 对应的设备文件是/dev/tpm，试以例 3.8 中的 TPM_Transmit 函数为基础，给出一个向 TPM 发 TPM_Reset()命令的 C 语言函数。

解答：可用于向 TPM 发 TPM_Reset()命令的函数 TPM_Reset 定义如下（行号是为了说明方便而加的）：

```
1.  uint32_t TPM_Reset()
2.  {
3.      unsigned char blob[4096] = {
4.          0,193,      /*TPM_TAG_RQU_COMMAND*/
5.          0,0,0,10,   /*blob 指向的包的长度，单位为字节 */
6.          0,0,0,90};  /*TPM_ORD_Reset */
7.      return(TPM_Transmit(blob));
8.  }
```

第 3～6 行构造出了相应的命令包，第 4 行描述包的类型，声明这是一个命令请求包，第 5 行说明包的长度为 10 字节，第 6 行描述命令的序号，TPM_Reset()命令的序号是 90。

例 3.9 中定义的数组 blob[]除了要表示命令包外，还用于接收响应包，所以，它的大小定义为响应包的最大长度（4096 字节）。

TPM_GetCapability()是另一个简单的 TPM 命令，它可以提供当前 TPM 的版本号、TPM 中的密钥槽总数、已装载的密钥数、已装载密钥的句柄和 PCR 寄存器数等。例 3.10 说明应用该命令读取 TPM 的版本号的方法。

例 3.10　设在 Linux 操作系统中，TPM 对应的设备文件是/dev/tpm，试以例 3.8 中的 TPM_Transmit 函数为基础，给出一个向 TPM 发 TPM_GetCapability()命令的 C 语言函数。

解答：可用于向 TPM 发 TPM_GetCapability()命令的函数 TPM_GetCapability_Version 定义如下（行号是为了说明方便而加的）：

```
1.  uint32_t TPM_GetCapability_Version()
2.  {
3.      unsigned char blob[4096] = {
4.          0,193,      /* TPM_TAG_RQU_COMMAND */
5.          0,0,0,18,   /* blob 指向的包的长度，单位为字节*/
```

```
6.              0,0,0,101,    /* TPM_ORD_GetCapability */
7.              0,0,0,6,      /* TCPA_CAP_VERSION */
8.              0,0,0,0};     /*  无须再取其他信息  */
9.          return(TPM_Transmit(blob));
10.  }
```

第 3～8 行构造出了相应的命令包，第 5 行说明包的长度为 18 字节，第 6 行给出的命令序号 101 对应的是 TPM_GetCapability()命令，第 7 行给出的参数指示要读取 TPM 的版本号。

3.7.5　TPM命令的基本用途

3.7.4 节着重介绍了 TPM 命令请求消息和命令响应消息的格式及它们在程序中的使用方法，通过对这些内容的学习，基本了解了在程序中如何让 TPM 去为我们工作。本节将从程序员的角度探讨在程序中可以利用 TPM 去完成什么样的工作。我们通过了解 TPM 提供的基本命令来寻找问题的答案。

对于一个新出厂的 TPM，必须先为它建立属主，它才能开始正常发挥作用。在 BIOS 将 TPM 启动并清零之后，可以使用 TPM_TakeOwnership()命令为该 TPM 建立属主，该命令完成以下几项关键任务：

（1）建立由属主提供的属主授权秘密；

（2）创建存储根密钥（SRK）；

（3）建立由属主提供的 SRK 授权密钥；

（4）将 SRK 的公钥返回给属主。

TPM 的某些命令的执行需要经过授权检查，授权检查成功才表明发出命令者有权执行该命令，该命令才得以执行。与属主有关的命令通常都需要根据 160 比特的属主授权秘密进行授权检查。类似地，密钥的使用可能需要根据密钥的授权秘密进行授权检查，密钥的授权秘密在创建密钥时建立。

由 3.6.4 小节可知，TPM 支持 OIAP 和 OSAP 两个授权检查协议，它们都建立授权环境和对应的句柄，把句柄返回给发命令实体，都使用即时随机数，不同的是，OIAP 建立一个长期的会话和对应的会话秘密，该会话秘密可以在同一个会话里的多个对象中使用，OSAP 只与一个对象关联，如与一个给定的密钥关联。执行 TPM_TakeOwnership()命令时，必须采用 OIAP，因为此时对象和秘密都还没建立。

TPM_OIAP()和 TPM_OSAP()都创建授权会话句柄，授权会话结束时，应该通过执行 TPM_Terminate_Handle()来释放相关的句柄。

TPM 提供的重要功能包括数字签名和加密/解密，密钥的生成是使用这些功能的关键一环。TPM_CreateWrapKey()命令借助硬件的随机数生成器在芯片上生成一对新的 RSA 密钥，密钥的类型可以是签名密钥或加密/解密密钥。TPM 不允许用签名密钥进行

加密/解密，也不允许用加密/解密进行签名。生成新密钥时，可以为该密钥指定一个秘密，这样，以后使用该密钥时，必须提供该秘密。还可以为新密钥指定一些 PCR 寄存器的值，在这种情况下，只有当 TPM 中的 PCR 寄存器值与指定给新密钥的 PCR 寄存器值匹配时，才能使用该密钥。所有的密钥都必须有一个父密钥（可以是 SRK），把一对密钥返回给用户之前，必须用父密钥对该密钥对的私钥进行加密。返回的密钥由用户保存，以后需要时，再装载使用。

TPM_LoadKey()命令用于把一个密钥装载到 TPM 的一个易失性密钥存储槽中，执行该命令时，要求用户提供相应密钥的父密钥的授权秘密（如口令），相应密钥装载后，TPM 用该父密钥对装载的密钥的私钥进行解密，以供使用。

TPM_Sign()命令使用已装载的密钥对给定的数据进行签名，给定的数据通常是实际数据的哈希值。

TPM_Seal()命令用于对数据进行 RSA 加密，它要求提供一个已装载的加密密钥及相应的密钥授权秘密（如果有的话）。使用该命令时，也可以指定一些 PCR 寄存器的值，这样，只有当 TPM 中的 PCR 寄存器值与加密时指定的 PCR 寄存器值匹配时，才能对加密数据进行解密。使用该命令时，还可以为加密数据指定数据授权秘密（如口令），这样，解密时，需要检查数据授权秘密。

对于用 TPM_Seal()命令加密的数据，可用 TPM_Unseal()命令进行解密。

TPM 提供的重要功能还包括完整性的度量和报告。完整性度量的结果存放在 PCR 寄存器中，那么，用什么命令对 PCR 寄存器进行操作呢？

TPM_Extend()命令用于把一个度量值加入到一个 PCR 寄存器中，如果给定的度量值为 inDigest，指定的寄存器编号为 pcrNum，则执行该命令的效果为

PCR[pcrNum] = SHA-1 [PCR[pcrNum] || inDigest]

TPM_PCRRead()命令把指定 PCR 寄存器的当前内容直接返回给发出该命令的实体，PCR 寄存器的内容是 20 字节（160 比特）的值。该命令可用于检查完整性度量值。

TPM_Quote()命令用于报告指定 PCR 寄存器的值的加密处理结果，该命令要求提供一个已装载的密钥和一个外部数据（通常是即时随机数），它用该密钥对描述 PCR 寄存器的当前值和外部数据的语句进行签名。如果已装载的密钥为 keyH，指定的寄存器编号为 pcrNum，外部数据为 extD，则该命令的返回结果可描述为

sig { PCR[pcrNum], extD } keyH

最后，简述 TPM 的真实身份及隐私问题。TPM 的真实身份由背书密钥（EK）标识，TPM_ReadPubek()命令可用于读取 EK 的公钥（Pubek）。用 TPM_TakeOwnership()命令为 TPM 建立属主时，需要用 Pubek 加密向 TPM 传送的敏感数据（即属主授权秘密）。为了保护隐私，建立了 TPM 的属主以后，属主应该禁用 Pubek，此后，执行 TPM_ReadPubek()命令将导致失败。

3.8　本章小结

可信计算组织（TCG）提出的可信计算体系以硬件安全芯片为基础来构建可信计算平台，以解决信息系统的安全问题。为了认识硬件在信息系统安全中的基本地位，本章以 TCG 的可信计算体系为代表，介绍可信计算平台的主要思想与基本技术，主要内容包括可信计算发展概貌、可信计算平台的基本特性、可信计算平台的基本体系、可信平台模块、可信计算平台的隐私问题、可信计算平台的运行模型及可信计算平台的编程接口等。

20 世纪 70 年代末，尼巴尔第对可信计算的概念进行了探讨，建立了可信计算基的思想。1999 年，可信计算平台联盟（TCPA）创立，2003 年，TCPA 演变为可信计算组织（TCG），促使可信计算向实用化推进了一大步。可信平台模块（TPM）是 TCG 可信计算的基础。

保护能力、对外证明、完整性度量/存储/报告等是可信计算平台的基础特征。保护能力是一组具有访问保护区域的特定权限的命令。对外证明是证明信息的正确性的过程，由 TPM 提供的对外证明用 TPM 的工作身份密钥（AIK）对 TPM 的内部数据进行签名，AIK 可以借助 TPM 的真实身份密钥（EK）来生成。完整性度量获取影响平台完整性的度量值并把它的摘要存放到 PCR 中。

可信计算平台通常拥有度量信任根（RTM）、存储信任根（RTS）和报告信任根（RTR）等信任根，可信构件块（TBB）和信任根的组合形成信任边界，信任可以从信任根开始不断向外传递。完整性度量行为可以用度量核、度量事件、被度量的值和度量产生的摘要等概念描述。可用 AIK 对完整性报告进行签名，认证 PCR 值的真实性，还可以把一个随机数组合到 PCR 的签名中，以防止重放攻击。

为了验证 AIK 关联的信任声明的有效性，需要使用凭证，主要凭证有：EK 凭证、符合性凭证、平台凭证、有效性凭证和 AIK 凭证等。保护信息交换的主要操作有绑定、签名、封装和封装的签名等。

RTS 保护密钥和数据的安全，管理少量的易失性存储器（密钥槽），用于在签名和解密时存放密钥。密钥经过加密后可以存储到 TPM 之外的存储器中。存储根密钥（SRK）用于保护存放在 TPM 的外部的其他密钥。

TPM 的主要成分包括输入/输出组件、非易失性存储器、PCR 寄存器、工作身份密钥、程序代码、随机数生成器、SHA-1 引擎、密钥生成、RSA 引擎、功能开关和执行引擎等。

TCG 遵循万维网联盟（W3C）隐私偏好平台（P3P）工作组的隐私指导原则建立可信计算平台的隐私模型，采用 P3P 的指导原则保护平台的身份信息。EK 标识平台的真实身份，它不用于对外证明，AIK 是 EK 的别名，反映平台的工作身份，用于实施对外

证明，从而避免泄露平台真实身份，保护平台隐私。

TCG 为 TPM 定义了多种运行模式（状态），以适应在多种应用场景中对 TPM 进行部署和维护。启用、激活和建立属主等方面的操作模式可以制约 TPM 的行为。平台从 RTM 开始启动，系统初始化操作对平台的组件和配置进行度量，操作系统帮助实施系统的完整性，平台可以对应用程序进行度量，应用程序也可以对平台进行检查。

TCG 可信计算平台定义了驱动程序库接口（TDDLI）、软件核心服务接口（TCSI）和服务提供方接口（TSPI）等三层软件接口。TPM 命令验证确定实体是否拥有执行命令的合法授权。TCG 为 TPM 命令验证定义 ADIP、ADCP、AACP、OIAP 和 OSAP 五个协议。

程序员视角的 TPM 可分为加密功能单元、非易失性存储器和易失性存储器三大部分。加密功能单元提供加密/解密和哈希运算支持。非易失性存储器用于存放 EK、SRK 和属主授权秘密。易失性存储器提供密钥槽、PCR 寄存器、密钥句柄和授权会话句柄。

TPM 命令接口采用“请求−响应”交互模型，命令以命令消息的形式在实体与 TPM 之间传递，命令的语法以按次序排列的方式表示。在程序设计中，可以把命令请求消息组织成命令包，把命令响应消息组织成响应包。向 TPM 发出命令，就是把命令包传给 TPM；从 TPM 获取响应结果，就是从 TPM 接收响应包。

TPM 命令的基本用途包括为 TPM 建立属主、建立授权环境、生成密钥、把密钥装载到密钥槽、对数据进行签名、对数据进行加密/解密、把度量值加入 PCR 寄存器、读取 PCR 寄存器内容、报告 PCR 寄存器的加密处理结果、读取 EK 的公钥等。



习题 3

1. 你所知道的可信计算方面最早的研究工作是在什么时候开展的？代表性人物是谁？代表性思想是什么？

2. TCG 所定义的“可信”主要指什么？根据该定义，试举例说明什么样的系统是可信的？如果一台计算机上有病毒，你认为它还可信吗？为什么？

3. 根据 TCG 规范，把一个现有的计算机系统改造成一个可信计算平台的基本方法是什么？试举例说明什么样的计算机可以改造成可信计算平台？

4. 一个可信计算平台应该具备哪些基本特性？它们的含义是什么？试举例说明。

5. 在可信计算平台中，信任根指的是什么？主要的信任根有哪些？

6. 如果要把一台个人计算机改造成一个可信计算平台，那么，个人计算机中的什么成分可以改造成度量的核心信任根（CRTM）？在该可信计算平台中，可信构件块（TBB）由哪些成分构成？

7. 在可信计算平台中，为什么要使用凭证？TCG 主要定义了哪些类型的凭证？

8. 在可信计算平台中，能够唯一地标识平台真实身份的是 EK，但是，为了保护平台的隐私，EK 不能暴露给外界，用于对外标识平台身份的是 AIK，试说明为什么根据

AIK 可以定位一个确定的平台？

9. TPM 中的 PCR 寄存器的主要作用是什么？保存在 PCR 寄存器中的值是采用什么计算公式计算出来的？

10. 存储信任根（RTS）与存储根密钥（SRK）之间是什么关系？试举例说明 SRK 的用途？

11. 在可信计算平台环境中，“把数据封装到平台上”是什么意思？试举例说明。这种数据封装有什么好处？有什么弊端？

12. 一个可信计算平台 A 向远端的请求方 B 证明平台的完整性的基本方法是把 A 中的指定 PCR 寄存器的值提供给 B，试问在这个过程中存在什么样的重放攻击危险？采取什么方法可以避免这种重放攻击？（提示：重放攻击亦称中间人攻击，指通信中存在第三者，导致接收方接收到的不是发送方发送的真实信息。）

13. 根据 TCG 的接口规范，为了利用可信硬件平台建立可信应用系统，需要开发哪些类型的软件组件？哪些软件组件将与系统安全密切相关？

14. TPM 命令的授权验证主要解决什么问题？相关的协议有哪些？其中需要显式确定的协议是哪些？试举例说明在实际应用中需要确定相关协议的场景。

15. 设有一个支持 TPM 的 Linux 系统，请设计一个可以在该系统中运行的 C 语言程序，要求该程序能够根据用户的需要对用户提供的信息进行加密/解密，并且，加密/解密操作要借助 TPM 硬件来完成。（说明：涉及 TPM 命令时，可以忽略参数的具体形式，只须表达基本原理。）

第二部分

核　心　篇



第 4 章
操作系统的基础安全性



操作系统是直接控制硬件工作的基础软件系统，它不但是计算机资源方便和有效使用的基础，也是信息系统安全的软件基础。本章首先概述操作系统安全性的整体情况，然后从身份认证、访问控制、特权管理、加密支持和安全审计等方面介绍操作系统的基础安全支持作用，主要内容包括身份标识与认证的基本方法、面向网络的身份认证、基于 PAM 的统一认证框架、基于权限位的访问控制、进程的有效身份与权限、基于 ACL 的访问控制、特权分割与访问控制、加密文件系统和系统行为审计等。

4.1　操作系统安全概貌

信息系统的安全性由硬件与软件的相互配合来提供支持。在传统意义上，硬件的安全机制和操作系统低层的安全机制称为保护机制。内存保护就是系统保护机制必须履行的典型职责之一。保护机制是高层安全机制和安全服务的基础。保护机制、安全机制和安全服务是操作系统安全性的关键。

操作系统安全性研究的历史源远流长。计算机系统的资源共享催生了原始的操作系统安全需求，而计算机系统在军事领域的应用则成为操作系统安全性研究的强大驱动力。

4.1.1　操作系统安全简史

操作系统安全性的基本理论、技术和方法是在 20 世纪 60～70 年代逐步建立起来的，此后，操作系统安全性的研究经历了持续不断的发展，今天，依然有很多问题有待于进一步探索和解决。

1946 年 2 月 14 日，世界上第一台电子计算机诞生在美国宾夕法尼亚大学，它的名字叫 ENIAC。20 世纪 50 年代中期，世界上第一个操作系统诞生在美国通用汽车研究实验室，那是一个为 IBM 701 主机配备的批处理系统。20 世纪 60 年代初期，世界上第一个分时操作系统 CTSS 诞生在美国麻省理工学院（MIT）计算中心。

分时操作系统诞生后，多个用户可以同时使用同一个计算机系统，从而使计算机系统资源的使用效率得到了提高，与此同时，也使共享资源的保护问题呈现在人们的面前。分时操作系统支持多个进程的并发执行。进程是在内存中执行的，内存就是典型的共享资源，即多个进程需要共享内存资源。共享内存资源的进程不能互相干扰，一个进程不能非法访问分配给另一个进程的内存空间。显然，保护内存空间是操作系统必须提供的基本安全支持。

20 世纪 60 年代中后期，在 CTSS 的基础上，美国的 MIT、贝尔实验室和通用电气公司联合开发了 Multics 操作系统。在 Multics 项目中，内存保护是一个要实现的重要目标，该项目探讨了硬件与软件相结合的内存保护解决方案。

隔离是解决共享资源保护问题的基本措施。把分配给进程 A 的内存空间与分配给进程 B 的内存空间隔离开，是防止这两个进程相互干扰的有效办法。利用虚拟机技术可以

有效地获得隔离的效果。20 世纪 60 年代中后期，IBM 系统研究与开发中心开发了 CP/CMS 操作系统，这是一个为 IBM 360 机器配备的基于虚拟机的分时系统，它能把一台 360 机器模拟成多台并发工作的 360 机器，各台虚拟的 360 机器可以运行不同的操作系统。20 世纪 70 年代初期，CP/CMS 发展成 VM/370，装备 IBM 370 机器。

在普通操作系统保护机制的基础上，军事应用对操作系统的安全性有特殊的要求，最突出的一个要求是敏感信息的分级处理。现实中的敏感信息需要划分为绝密、机密、秘密和非密等多个等级，军事应用要求操作系统具有与此相适应的信息处理能力。

1967 年，美国国防部组建计算机安全特别机构，推动面向军事应用需要的安全操作系统建设工作。20 世纪 60 年代中后期诞生的 Adept-50 可以说是历史上第一个安全操作系统，这是一个运行在 IBM 360 机器上的分时系统，它提供用户和信息的安全等级标记功能，以形式化的高水标安全模型为基础，实现了美国的一个军事安全系统模型。

20 世纪 60 年代后期，B.W. Lampson 通过主体、客体和访问矩阵等概念建立了形式化的访问控制模型，为操作系统访问控制机制的开发奠定了理论基础。20 世纪 70 年代初期，D.E. Bell 和 L.J. Lapadula 建立了第一个可证明的安全模型，即 Bell & Lapadula 模型，简称 BLP 模型。该模型依据主体和客体的安全等级标记来定义访问控制规则，是多级安全系统的理论基础，对多级安全操作系统的研究与开发具有重大影响。

在安全系统的结构设计与开发方法方面，1972 年，在美国空军一项计算机安全规划的研究成果中，J.P. Anderson 提出了引用监控机、引用验证机制、安全核和安全建模等重要思想。1975 年，J.H. Saltzer 和 M.D. Schroeder 总结了保护机制的体系结构并提出了安全机制设计的八大原则：经济性原则、失败−保险原则、完全仲裁原则、开放设计原则、特权分离原则、最小特权原则、最少公共机制原则及心理可接受性原则。这些对操作系统安全机制的设计与开发具有重要的影响。

1983 年，美国国防部颁布了世界上第一个计算机安全评价标准，即可信计算机系统评价标准，简称 TCSEC。TCSEC 文本的封面是橘色的，因此，人们把它称为橘皮书。1985 年，美国国防部颁布了 TCSEC 的修订版。TCSEC 颁布之后，与之相配套的一系列规范不断问世，每一种规范文本的封面都呈现不同的颜色，因此，人们把 TCSEC 标准及与其相配套的这一系列规范统称为彩虹系列。

在 TCSEC 标准颁布后不到十年的时间里，彩虹系列在安全操作系统领域产生了巨大的影响，大量的研究与开发工作纷纷把 TCSEC 标准的安全评价等级设定为所追求的目标，一系列安全操作系统应运而生，这为操作系统安全性的研究创造了难得的财富。

TCSEC 标准的特征是安全策略的单一性，随着 20 世纪 90 年代初 Internet 影响的迅速扩大及分布式应用的迅速普及，单一安全策略框架的局限性日益明显。1992 年，美国推出联邦标准草案，欲以之取代 TCSEC 标准，以消除 TCSEC 的局限性。1993 年，美国国防部在 TAFIM 计划中推出了称为 DGSA 的新的安全体系结构，其显著特点之一是要求为多种安全策略提供支持。1999 年，CC 标准成为信息安全评价的国际标准，它在

一定意义上体现了信息安全系统安全特性的灵活性和多样性。

随着技术的进步和应用的发展，在开放式、分布式的应用环境中，实现对安全策略多样性的支持逐渐成为操作系统安全性研究追求的目标。

4.1.2　操作系统安全的主要内容

对于操作系统的安全性，可以站在操作系统使用者的角度来学习，也可以站在操作系统开发者的角度去分析，可以从主流通用操作系统的侧面来考察，也可以从专用安全操作系统的侧面去剖析。

与操作系统安全性相关的内容非常丰富，在有限的篇幅里很难一一列全，以下是一些主要部分。

（1）内存保护：在多用户、多任务的系统中，实现内存空间的隔离与共享，防止不同进程间非法访问或破坏；

（2）客体重用：在存储介质的分配中，防止遗留信息的泄露；

（3）身份标识与认证：标识用户的有效身份，认证用户使用系统的合法性；

（4）访问控制：控制主体访问客体的行为，防止出现非法访问行为，根据需要，提供自主访问控制或强制访问控制；

（5）可信路径：实现用户与可信软件之间、可信软件与可信软件之间的可信通信；

（6）加密支持：提供信息加密和解密及密钥管理的有效支持；

（7）特权管理：实现特权的合理划分、使用和管理，以便支持最小特权原则；

（8）安全审计：记录、管理和报告安全相关行为信息，为安全行为分析提供支持；

（9）多安全策略的灵活支持：根据应用需要，支持多种安全策略，支持安全策略的灵活选择和配置；

（10）隐蔽信道处理：防止或限制利用非法隐蔽通信途径泄露信息；

（11）完整性与可用性保护：防止系统或数据遭受非法篡改，提供系统失效后恢复正常工作的能力。

4.1.3　必不可少的操作系统安全性

操作系统是离硬件最近的、介于硬件与应用软件之间的基础软件系统，这样的特殊地位决定了操作系统的安全性在确保信息系统的安全性中具有不可替代的作用，主要体现在：如果没有操作系统提供的安全支持，应用系统的安全性是无法得到保障的。操作系统提供的安全支持很多，仅从强制安全性和可信路径两个侧面，就足以说明这样的问题。

1．强制安全性及其作用

什么是强制安全性？美国国防部的橘皮书把强制安全性与国防部的多级安全性策略密切地联系在一起，给出了强制安全性的如下传统定义。

定义 4.1　强制安全性就是以美国国防部的多级安全性策略为基准的强制访问控制特性。

然而，这样的定义忽视了诸如非传递性、动态职责分离等一些非常关键的特性，既满足不了美国国防部的要求，也满足不了其他领域的要求。从一般意义上说，一个安全性策略是不是强制安全性策略，要视其策略逻辑的定义与安全属性的分配的方式，因此，我们有如下定义。

定义 4.2　如果一个安全性策略的策略逻辑的定义与安全属性的分配只能由系统安全性策略管理员进行控制，则称该安全性策略为强制安全性策略。

强制安全性能够实现整个组织全范围的安全性策略，也称为非自主安全性。

与强制安全性相对应的是自主安全性，它体现普通用户在策略逻辑定义和安全属性分配方面的自主特点。

定义 4.3　如果普通用户能够参与一个安全性策略的策略逻辑的定义或安全属性的分配，则该安全性策略称为自主安全性策略。

一个操作系统的强制安全性策略可以分解成多种具体的安全策略，如强制访问控制策略、强制认证支持策略、强制加密支持策略。强制访问控制策略规定主体在操作系统的控制下如何对客体进行访问。强制认证支持策略规定必须采用什么认证机制对进入系统的实体进行认证。强制加密支持策略规定必须采用什么加密机制对数据进行保护。

一个安全系统必须提供一个对强制安全性策略进行定义和转换的架构，用于定义操作系统的强制安全性策略，以及将其转换成底层的操作系统强制安全性机制能够解释的形式。没有这样的架构就不可能保证强制安全性机制能够提供所需的安全特性。

只要强制安全性策略涉及机密性因素，在一个提供强制安全性的操作系统中就可能出现高带宽的隐蔽信道。不过，这不应该成为拒绝接受强制安全性的理由。就算可能会有隐蔽信道出现，通过提高对手对付我们的系统所需要面对的工作的复杂度，具有基本强制安全性控制的操作系统也能够增强系统的安全性。具有基本强制安全性控制的系统一旦成为主流系统，隐蔽信道被利用的现象将变得更加普遍，人们对解决系统中的隐蔽信道问题的关注程度也会进一步提高。

在支持强制安全性的系统中，为了执行一些安全相关的功能，有些应用程序需要拥有强制安全性策略中的特权，这样的应用程序通常被称为可信应用程序，因为我们对它们正确执行安全相关的功能给予了信任，并且相信它们不会滥用所拥有的特权。如果安全操作系统的强制安全性机制只支持粗粒度的特权，整个系统的安全性就落到了系统中的可信应用程序的安全性之上。为了降低对可信应用程序的依赖程度，操作系统强制安全性机制的设计应支持最小特权原则。

操作系统的强制安全性机制可用于严格地确保操作系统中的子系统不会被旁路并不会被篡改，进而支持应用程序的安全相关功能。例如，强制安全性机制可用于实现可信信息通道，可信信息通道能够确保从指定的信息源流向指定的目的地的信息，必须流经一个安全相关子系统，同时，能够确保相应的子系统的完整性。这些应用程序的很多安全需求可以由底层操作系统的强制安全性机制给予确保。

操作系统的强制安全性机制也可以用于把一个应用程序严格地限制在一个单一的安全域中，明确地与系统中的其他安全域分离开。在这种情况下，虽然应用程序中仍然可能出现不良行为，但可能导致的破坏作用已被限制在一个单一的安全域中。这种限制特性对于支持某种安全性策略的数据流控制是至关重要的。除了支持非可信软件的安全运行之外，这种限制特性还可以提供功能方面的支持。例如，提供一个独立的测试环境，或者提供一个隔离的开发环境。

虽然人们可以试图通过自主安全性机制实现强制安全性策略，但是，这种方案不能防御不经心的用户或恶意的用户。由于自主安全性机制把确定安全性的重任放在用户身上，任何一个用户在任何一个时间点上的不谨慎都有可能导致违反强制安全性策略的情况发生。相反，强制安全性机制把确定安全性的责任限制在系统安全策略管理员身上。在只有自主安全性机制的情况下，可以访问敏感性数据和应用程序的恶意用户可以直接透露敏感性信息，违反强制安全性策略。虽然同样的用户也可以不借助计算机系统就泄露信息，但通过计算机系统泄露信息可以提高泄露的带宽，可以降低泄露的可追查性。相反，在有强制安全性机制的情况下，用户只能通过隐蔽信道泄露敏感性信息，带宽受到了限制，对隐蔽信道的审计也提高了泄露的可追究性。

就算所有用户都是善意的且谨慎的，在只有自主安全性机制的情况下，有缺陷的或恶意的应用程序也可能破坏强制安全性策略。应用程序不一定能适时地启用强制安全性策略要求的安全机制，或者使用安全机制的方式不一定能与用户的意图一致。强制安全性机制能够保证及时地启用安全机制，并防止用户无意中执行非可信赖的应用程序。用户可能已经小心谨慎地定义了用于实现强制安全性策略的自主安全性策略，但应用程序可能未经过用户许可或未让用户知晓就改变自主安全性策略。相反，强制安全性策略只能由系统安全策略管理员调整。

在个人计算机系统中，用户可能同时是系统安全策略管理员，就算是这种情形，强制安全性机制也有助于防御有缺陷的或恶意的软件的不良行为。在最简单的情形中，系统只区分普通用户和系统安全策略管理员两种角色，这时，强制安全性机制能够防止用户不经意地执行非可信赖软件。如果根据职责的不同，把普通用户角色进一步细分成多种角色，强制安全性机制还能限制有缺陷的或恶意的软件可能导致的破坏的范围。

2．可信路径

定义 4.4　可信路径是一种实现用户与可信软件之间的直接交互作用的机制，它只

能由用户或可信软件激活，不能由其他软件仿冒。

如果没有可信路径机制，恶意软件就可以在用户面前冒充可信软件或在可信软件面前冒充用户。潜在地，这样的恶意软件有可能获取到敏感性信息，以用户的名义执行违反用户意图的功能，或者给用户建立一个某项功能已经启动的假象，而实际上该项功能根本没有启动。除了支持基本系统中的可信软件外，可信路径机制应该具有可扩展能力，支持系统安全策略管理员根据需要添加可信应用程序。

可以进一步拓宽可信路径的思想，使交互作用的范围不局限于可信软件与用户之间。可信通道的概念就是为了解决不同网络组件上的可信软件之间的通信问题而提出的。更加一般化的一个概念是保护路径，它是为了保证关键系统功能不被假冒而提出的，它是确保两个对象之间的相互认证渠道的一种机制。虽然局部通信中的保护路径机制可以在应用空间中建立，可以不用操作系统的直接认证支持，但是，操作系统最好提供自己的保护路径机制，因为这样的机制更容易得到保障，并且，可能更加有效。

3．访问控制机制的后盾

应用空间中的一个访问控制机制可以分解成一个实施组件和一个判定组件。当主体试图访问受访问控制机制保护的客体时，实施组件必须启动判定组件，向判定组件提供合适的输入参数，然后实施由判定组件返回的结果。输入参数的常见例子是主体和客体的安全属性。为了执行策略判定，判定组件可能还要查询其他外部资源，例如，它可能需要使用外部策略数据库和当前时间等系统信息。

如果一个恶意实体能够篡改或访问控制机制中的任何一个组件或任何输入参数，它就能够破坏相应的访问控制机制。就算所有的组件和输入参数都存放在一个单一的文件中，我们也必须依靠操作系统的安全性机制来保护相应文件的完整性。只有强制安全性机制才能严格地提供这样的完整性保证。

就算用于安全判定的所有输入都得到了强有力的保证，如果一个授权用户启动了恶意软件，恶意软件也就可以未经用户同意或在用户毫不知情的情况下更改客体的安全属性或策略数据库的规则。所以，为了保证在没有得到用户明确授权的情况下访问权限不随意扩散，访问控制机制需要操作系统的可信路径机制提供支持。

如果一个恶意实体能够在实施组件面前冒充判定组件，或者在判定组件面前冒充某输入源，那么，该恶意实体也就能够破坏相应的访问控制机制。如果访问控制机制的结构组件或输入源并非全部存放在同一个单一的应用程序中，那么访问控制机制就需要保护路径机制的支持。

如果一个恶意实体能够躲避实施组件的控制，那么它也就能够轻而易举地破坏访问控制机制的作用。操作系统的强制安全性机制可保证所有对受保护客体的访问都必须通过实施组件的仲裁。

4．加密保护机制的后盾

关于应用空间的加密保护问题，要从加密机制自身及加密机制的启动两个方面进行分析。

首先，假设加密机制是一个硬件组件，它能够正确实现所需的加密功能，并且，可以通过安全的方法在硬件组件中建立加密密钥。在这样的假设前提下，不需要操作系统的支持就能保证算法和密钥的机密性和完整性。在这样的硬件化程度相当高的加密机制中，依然存在着只有安全操作系统的功能才能够有效地处理的弱点。

第一个弱点是加密机制的启动得不到保证。恶意应用程序或恶意用户有可能会绕过或改动启动加密机制的应用程序，恶意应用程序有可能在执行启动操作的应用程序面前冒充加密机制，因而，合法使用加密机制的尝试不一定能把加密机制启动起来。

操作系统的强制安全性和保护路径特性能处理这个弱点。强制安全性机制能够保证启动加密机制的应用程序不被旁路，并且不被恶意软件或恶意用户更改。通过内嵌加密机制的方法，即物理上把加密机制插入到数据发送方和数据接收方的中间，也可以实现不可旁路性，但这种方法很不灵活。保护路径机制能够防止恶意软件在执行启动操作的应用程序面前冒充加密机制。

第二个弱点是加密机制被滥用，滥用包括非授权应用程序对服务、算法、会话或密钥的使用。如果没有操作系统支持对调用方进行身份标识，只要有一个用户激活了加密机制，系统中的任何应用程序就都可以使用授权给该用户使用的任何服务、算法、会话或密钥，加密机制本身无法对不同的使用者加以任何区分。在这种情形下，加密机制可能被为其他用户工作的应用程序滥用，或者被以授权用户名义工作的恶意软件滥用。而且，除非加密机制为用户执行启动操作提供直接的物理接口，否则，恶意软件可以在用户面前冒充加密机制，骗取认证信息，然后在未经用户同意或用户毫不知情的情况下激活加密机制。即使是为用户提供直接的物理接口，加密机制要求用户每执行一次加密操作都进行确认也是不现实的。

第二个弱点可以通过操作系统的强制安全性、可信路径和保护路径特性来处理。可信路径机制消除了提供独立的物理启动接口的必要。保护路径机制使加密机制能够对调用方进行身份标识，并对服务、算法、会话和密钥的使用实施细粒度控制。可以不需要加密机制处理细粒度用法的控制，而让强制安全性机制提供这种控制。例如，强制安全性机制可以把加密机制隔离起来，只允许激活该机制的用户所运行的应用程序使用。

所以，即使是最简单的情形，安全操作系统的特性对于解决应用空间中加密保护的弱点的问题也是至关重要的。

如果取消初始密钥已安全地建立在加密机制中的假设，那么，新的弱点又出现了，初始密钥可能被非授权实体辨认或更改。除非通过与加密机制连接的专用物理接口提供初始密钥，否则，操作系统必须保护初始密钥源与加密机制之间的路径，并且保护初始

密钥源本身。强制安全性机制可以严格地保护相应的路径和密钥源，可信路径也可以为初始密钥的生成提供支持。

如果取消在一个单一的硬件组件中实现加密机制的假设，改用软件实现加密机制，那么，加密机制的代码和数据的机密性和完整性都要依靠操作系统来支持，包括内存保护和文件保护方面的支持。加密机制的完整性和机密性，必须由强制安全性提供严格的保证。如果加密机制要用到外部输入，如给随机数发生器提供的输入参数，那么，必须由强制安全性机制保护输入源及输入源与加密机制之间的路径。

5．操作系统安全的不可替代性

访问控制和加密保护是解决信息系统安全问题的两个重要的基本手段。以上的分析表明，没有操作系统对强制安全性和可信路径的支持，应用空间中的访问控制机制和加密保护机制都是不可能有效地实现的。所以，操作系统的安全性在信息系统安全中具有不可替代的作用。

4.2　身份标识与认证的基本方法

保障合法用户方便、有效地使用计算机系统并防止恶意用户非法干扰计算机系统是操作系统要实现的目标，所以，在很多情形下，操作系统的安全控制与用户的身份有密切的关系，合法用户身份的标识和认证是操作系统安全控制的基础。

4.2.1　身份标识的基本方法

为了区分合法用户和非法用户，操作系统可以对所有合法用户登记造册，花名册上可以找到的用户就是合法用户，花名册上找不到的用户可视为非法用户。

定义 4.5　为用户建立能够确定其身份状况的信息的过程称为对用户进行身份标识。

通常，操作系统为每个合法用户设立一个账户，为每个账户设立一组管理信息，以实现对系统合法用户的标识和管理。系统中所有用户账户的管理信息构成一个用户账户信息数据库。

操作系统使用账户名来建立用户与账户之间的连接。账户名是用户可以用来向操作系统表明自己的身份的名称，由用户根据自己的喜好来确定。习惯上，账户名用以字母开头的由字母、数字和下划线等字符组成的字符串来表示，如 wenchang、shi_ruc 和 wshi2008 等都是常见的账户名形式。操作系统要求账户名具有唯一性，不能出现重名，即一个操作系统中的两个不同用户不能使用相同的账户名。

账户名是用户身份的外部表现形式，主要供用户使用，方便用户记忆和使用也许可以成为确定账户名的原则之一。在操作系统的内部处理中，不直接使用账户名来标识用户的身份。根据内部处理的需要，操作系统自动为用户生成内部形式的身份标识（ID），并

建立账户名与内部身份标识之间的对应关系。最简单的用户身份内部表示法就是采用整数值作为身份标识号，如与账户名 wenchang 对应的身份标识号可能是 500。

例 4.1　在一个操作系统中，某用户的账户名和身份标识号分别是 wenchang 和 500，请问：它们是如何确定的，分别用于什么场合？

解答：账户名 wenchang 是由用户确定的，用户用它来向系统表明自己的身份，身份标识号 500 是由操作系统自动生成的，操作系统在内部处理中用它来标识用户的身份。

对用户进行分组是操作系统支持的用户管理中的一种常见方法，即多个用户可以组成一个用户组。与用户的表示方法类似，操作系统使用组名和组标识号来表示用户组，组名主要供用户使用，而组标识号主要供操作系统在内部处理中使用。例如，siselab_ms 和 siselab_phd 可以是两个组名，而 301 和 302 可以是它们分别对应的组标识号。操作系统也维护一个用户组管理信息数据库。

在用户界面信息中，操作系统一般显示用户名和组名，在内部信息处理中，操作系统一般使用用户标识号和组标识号。用户标识号和组标识号分别简称为 UID 和 GID。

用户账户信息数据库可以用普通文本文件来表示，文本文件中的一行可以表示一个账户的相关信息。图 4.1 是 UNIX 操作系统中由/etc/passwd 文件表示的账户信息数据库的部分信息示例。

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
wenchang:x:500:300:Wenchang Shi:/home/wenchang:/bin/bash
weinan:x:501:301:Weinan Li:/home/weinan:/bin/bash
xin08:x:520:302:Xin Wu:/home/xin08:/bin/csh
.....
```

图 4.1　UNIX 操作系统的账户信息文件/etc/passwd

账户信息文件/etc/passwd 的每一行表示一个记录，对应一个账户，每个记录的信息被冒号（:）划分成若干个字段，以 wenchang 账户为例，各字段信息表示的含义如下。

- wenchang：账户名。
- x：与该账户对应的口令的相关信息。
- 500：用户身份标识号。
- 300：用户所属用户组的组标识号。
- Wenchang Shi：有关该账户的注释。
- /home/wenchang：用户使用系统时的默认工作目录。

- /bin/bash：用户登录系统时，系统自动为其启动的程序。

类似地，用户组信息数据库也可以用普通文本文件来表示，文本文件中的一行可以表示一个组的相关信息。图 4.2 是 UNIX 操作系统中由/etc/group 文件表示的用户组信息数据库的部分信息示例。

```
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
sisefellow:x:300:wenchang,binliang,zhiyong,zhaohui
siselab_ms:x:301:weinan,hanchao,kankan,xiaoli,liuxing
siselab_phd:x:302:xin08,hongwei,yanjun
.....
```

图 4.2　UNIX 操作系统的用户组信息文件/etc/group

用户组信息文件/etc/group 的每一行表示一个记录，对应一个用户组，每个记录的信息被冒号（:）划分成若干个字段，以 sisefellow 用户组为例，各字段信息表示的含义如下。

- sisefellow：用户组的组名。
- x：与该组对应的口令相关的信息。
- 300：用户组的组标识号。
- wenchang，binliang，zhiyong，zhaohui：该用户组所包含的用户的对应账户名。

由图 4.2 可知，一个组通常包含多个账户，如 sys 组包含 root、bin 和 adm 等账户；一个账户也可以属于多个组，如 root 账户属于 bin、daemon、sys 和 adm 等组。

4.2.2　身份认证的基本方法

定义 4.6　确认用户的合法身份的过程称为对用户进行身份认证。

当一个用户欲以 wenchang 的身份使用操作系统时，操作系统通过用户身份认证机制判断该用户到底是否真的是 wenchang 用户。如果判断结果是肯定的，操作系统就允许其进入系统；如果判断结果是否定的，操作系统则拒绝其进入系统。

最简单且最常用的用户身份认证方法就是利用口令进行认证。口令也称为密码，通常是一个字符串，属于保密信息，应该只有用户本人和操作系统才能掌握该信息。用户必须首先通过身份认证，才能开始使用操作系统，身份认证工作一般在用户登录进入操作系统的过程中启动。在登录过程中，用户需要提供账户名和口令信息，操作系统验证这两项信息的合法性，如果合法，则认证通过；否则认证失败。

执行登录过程时，操作系统首先根据用户提供的账户名在账户信息数据库中进行检索，如果能检索到对应的账户信息，则表明账户名是合法的，登录过程可以继续；接着，操作系统接收用户提供的口令，并检查该口令信息与系统中记录的口令信息是否一

致，如果一致，则身份认证成功，用户可以进入操作系统的正常工作环境，使用操作系统提供的服务。

图 4.3 是通过 Linux 系统的经典登录界面登录的一个例子。

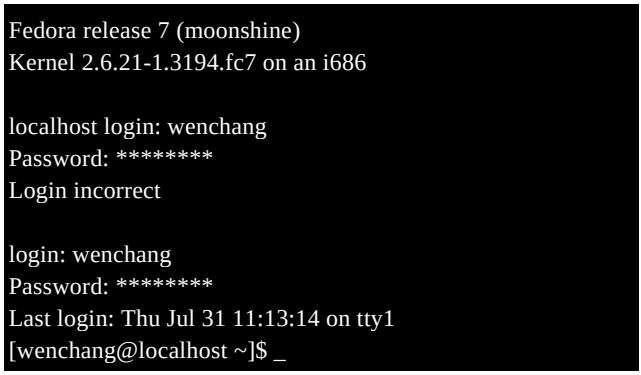


图 4.3　Linux 操作系统的登录过程

在这个例子中，系统给出 login 提示符时，用户输入 wenchang 账户名，系统给出 Password 提示符时，用户输入错误的口令，系统验证时发现口令不正确，身份认证不成功，因此登录失败。在第二次登录过程中，用户输入合法的用户名和正确的口令，身份认证成功，用户成功进入系统。用户输入口令时，系统是不显示口令的。

用户的口令是由用户自己确定和保管的，必要时，用户可以修改自己的口令。图 4.4 是用户在 Linux 操作系统中修改口令的例子。

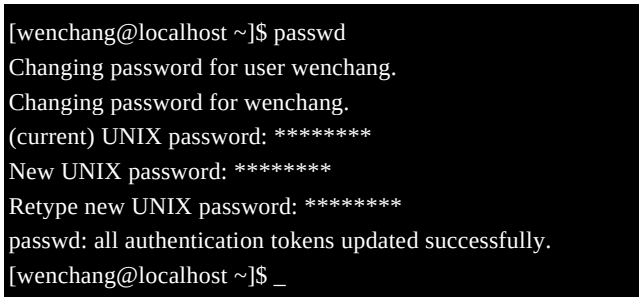


图 4.4　用户在 Linux 操作系统中修改口令

在图 4.4 中，第 1 行中的 passwd 是用户输入的修改口令的命令，第 2、3 行是描述命令功能的信息，第 4 行提示用户输入现有口令，第 5 行提示用户输入新口令，第 6 行提示用户再次输入新口令，第 7 行说明口令修改成功，第 8 行表示口令修改完毕，回到正常操作状态。

修改口令时，一般都要求用户首先输入现有口令，只有在用户输入了现有的正确口令后，系统才允许用户设置新的口令，新口令需要输入两次，以避免输入有误。

4.2.3　口令信息的管理方法

采用基于口令的身份认证方法，口令的保护是安全的关键。用户要保管好自己的口令，操作系统更要保护好用户的口令，以防止非法用户利用系统中保存的口令相关信息破获合法用户的口令。

1．口令信息的维护与运用

回顾图 4.1 中由/etc/passwd 文件表示的账户信息数据库，账户信息记录的第 2 个字段对应的是口令相关的信息，图中用 x 标注，不妨把该字段称为口令字段。

以下方法可以管理口令信息。

方法一：在口令字段中保存口令的明文，身份认证时，直接取其值与用户输入的口令进行对比。

方法二：用确定的算法对口令进行加密，在口令字段中保存口令的密文，身份认证时，将口令的密文进行解密，将解密后的口令与用户输入的口令进行对比。

方法三：用确定的算法借助口令进行某种运算，在口令字段中保存运算的结果，身份认证时，借助用户输入的口令进行相同的运算，将口令字段中保存的结果与运算结果进行对比。

方法一和方法二分别保存明文和密文形式的口令。方法一很难保证口令的安全，攻击者只要得到账户信息数据库文件，就获得了所有账户的口令。方法二难以抵抗口令猜测攻击，攻击者完全可能采取已知密文猜测明文的方法去破解口令。只有方法三值得进一步考虑，它在整个系统中不保存任何形式的口令。

例 4.2　请给出一个采用方法三在操作系统中管理用户口令信息的方案。

解答：选择一个分组密码算法 A_{crypt}，取一个由 64 位的全 0 的二进制位串构成的数据块 D_{p}，算法 A_{crypt} 可以利用密钥 K 把数据块 D_{p} 加密成结果数据块 D_{c}，即

$$D_{\text{c}} = A_{\text{crypt}}(K, D_{\text{p}})$$

设计一个算法 A_{trans}，算法 A_{trans} 可以把数据块 D_{c} 变换为一个字符串 s，即

$$s = A_{\text{trans}}(D_{\text{c}})$$

为用户创建账户时，取用户的口令为密钥 K，用 A_{crypt} 把 D_{p} 加密成 D_{c}，用 A_{trans} 把 D_{c} 变换成 s，在账户信息数据库的账户信息记录的口令字段中保存字符串 s。

进行身份认证时，根据用户输入的口令，用同样的方法生成字符串 s_{r}，从账户信息数据库中查找出 s，比较 s_{r} 和 s，如果两者相同，则认证成功；否则认证失败。

例 4.2 中提到了分组密码算法，我们简单解释一下这个概念。在密码学中，有流密码算法（Stream Cipher）和分组密码算法（Block Cipher），它们的定义如下。

定义 4.7　通过逐位加密的方式对信息进行加密的算法称为流密码算法；通过逐块

加密的方式对信息进行加密的算法称为分组密码算法。

一个信息块由多个比特位组成，使用分组密码算法时，首先要对信息进行分组，每组包含多个比特位，每组信息构成一个信息块。

例 4.2 给出的口令信息管理方案选择用户的口令作为密钥，用一个确定的算法 A_{crypt} 对一段确定的信息 D_p 进行加密，然后用一个确定的算法 A_{trans} 把加密结果变换成一个字符串 s，在系统中只保存变换得到的字符串 s，不保存用户口令。该方案的关键是根据用户的口令生成字符串 s。

例 4.3　请给出一个能把任意的 64 位的二进制位串变换成一个字符串的方法。

解答：以 6 位为单位，从左至右，对 64 位的位串进行分组，可以分出 10 个位组，最后剩余 4 位。可以把最后一个位组的最后 2 位与剩余的这 4 位合在一起看成一个 6 位的位组，这样，一共得到 11 个位组。把每个位组看成一个无符号二进制整数，整数的取值在 0～63 之间。

另外，选取以下 64 个字符：

“.”，“/”，0～9，A～Z，a～z

按照以上顺序，从左至右，确定这些字符的序号（起始序号为 0），即，“.”为 0 号，“/”为 1 号，“0”为 2 号，…，“z”为 63 号。

最后，遵照以上约定，采取以下方法，便可以把任意给定的一个 64 位的二进制位串变换为一个字符串：

（1）把给定的 64 位的位串划分成 11 个 6 位的位组；

（2）把每个位组变换为相应二进制数的值，得到 11 个整数；

（3）把每个整数变换为以其为序号的字符，得到 11 个字符；

（4）把 11 个字符按照相应位串的顺序排列，组成一个字符串。

例 4.3 给出的方法可以作为例 4.2 中要使用的 A_{trans} 变换算法。显然，如果知道算法 A_{trans} 和字符串 s，反推位串 D_c 并不困难，所以，用算法 A_{trans} 对位串 D_c 做变换主要是为了信息维护的方便，增强不了多少安全性。要增强安全性，必须加大根据 D_c、D_p 和算法 A_{crypt} 猜测用户口令（即密钥 K）的难度。

例 4.4　对于例 4.2 给出的口令信息管理方案，改进账户信息数据库中的口令字段信息的生成方法，可以提升借助口令字段信息猜测口令的难度，从而提高口令信息管理方案的安全性。请给出一个有助于改进口令信息管理方案安全性的口令字段信息生成算法。

解答：选择一个分组密码算法 A_{crypt}，它可以利用密钥 K 把 64 位的数据块 D_p 加密成 64 位的结果数据块 D_c，即

$$D_c = A_{crypt}(K, D_p)$$

根据例 4.3 的方法设计一个算法 A_{trans}，它可以把 64 位的数据块 D_c 变换为一个字符

串 s，即

$$s = A_{\text{trans}}(D_{\text{c}})$$

通过以下操作生成口令字段信息：

（1）用一个 64 位的全 0 的二进制位串构造一个数据块 D_{p}；

（2）$K =$ 用户的口令，$i = 0$；

（3）$D_{\text{c}} = A_{\text{crypt}}(K, D_{\text{p}})$；

（4）$D_{\text{p}} = D_{\text{c}}$，$i = i + 1$；

（5）如果 $i < 25$，则回到第（3）步；

（6）$s = A_{\text{trans}}(D_{\text{c}})$，取 s 为口令字段信息。

例 4.4 主要是通过重复多次加密的方法来提高安全性的，它共进行了 25 次加密。传统的 UNIX 操作系统就是采用这样的方案来管理口令信息的，这些操作系统内部实现了一个 crypt()算法，这是一个以 DES 为基础的算法，它实现这里的 A_{crypt} 算法的功能。

2．口令管理中的撒盐措施

先来看一个日常生活中的简单实验。假设有两杯完全一样的液体，如果随意地手工抓取两把盐，分别撒入这两杯液体中，那么，它们显然就变得不一样了，因为向它们撒入的盐是随意的，撒入到两杯液体中的盐的量很难完全一样。可想而知，仅凭现在的两杯液体，要想推知原来的液体的状况，肯定不是一件容易的事情。

撒盐的效应可以借鉴到操作系统的口令信息管理之中，只不过，日常生活中的盐（Salt）变成了操作系统中的随机数。

定义 4.8　在操作系统的口令管理中，给口令拌入随机数的过程称为给口令撒盐（Salting），拌入的随机数称为口令的盐值（Salt Value）。

同一个口令，拌入不同的盐值，就得到不同的结果。凭借撒盐后的结果，推测原来的口令，是有一定难度的，这就是撒盐的效应。

基于口令的身份认证方法的最大弱点是用户确定的口令往往比较容易猜测。给口令撒盐能增加口令的随机性，其主要目的是要提高破解口令的难度。通常，撒盐的基本方法是给原始口令加入盐值，再用加盐后得到的结果进行加密运算，将加密运算后得到的结果作为要保存和使用的口令信息。

在字典攻击中，攻击者可以根据口令信息的加密算法，事先计算出与所有可能的口令对应的加密结果，然后，将用户口令对应的加密结果与事先计算的结果进行对比，如果找得到相同的结果，便可破获用户的口令。实施撒盐措施后，一个口令可以对应大量的加密结果，从而大大增强了如字典攻击等攻击的口令破解难度。

定理 4.1　给口令撒盐可以提高破解口令的难度。

例 4.5　请举出给口令撒盐的一种实现方法。

解答：确定一个随机数生成算法 A_{random}，确定一个哈希算法 A_{hash}，对任意给定的用户口令 D_{pw}，通过以下操作对它进行处理：

（1）生成一个随机数：$D_{salt} = A_{random}()$；

（2）把随机数附加到口令上：$D_{tmp} = D_{pw} \| D_{salt}$；

（3）生成哈希值：$D_{hash} = A_{hash}(D_{tmp})$；

（4）把 D_{hash} 和 D_{salt} 保存到系统中，作为用户的口令信息。

其中，第（2）步中的“||”表示连接运算，即，把第二个操作数连接到第一个操作数的尾部。例如：

“abcd” || “ABCD”　→　“abcdABCD”

11000011 || 00111100　→　1100001100111100

在这个例子中，D_{salt} 是盐值，给口令 D_{pw} 撒盐后得到的结果是 D_{tmp}，最后生成的口令信息是 D_{hash}。这是典型的口令撒盐处理方法。

例 4.6　请给出一个在操作系统中进行用户口令信息管理和身份认证的方案，要求实施撒盐措施，说明主要算法的基本思想。

解答：这里给出的方案主要包含口令管理信息生成、账户信息数据库的口令信息维护和身份认证等三个部分。

第一部分：口令管理信息的生成，由算法 A_{gen} 实现，它以用户口令 D_{pw} 和盐值 D_{salt} 为输入，输出口令信息字符串 s，即

$$s = A_{gen}(D_{salt}, D_{pw})$$

首先，确定一个口令撒盐算法 A_{salt}、一个分组密码算法 A_{crypt} 和一个位串变换算法 A_{trans}。算法 A_{salt} 把盐值 D_{salt} 加入用户口令 D_{pw} 中，输出加盐后的结果。算法 A_{crypt} 利用密钥 K 把 64 位的数据块 D_p 加密成 64 位的结果数据块 D_c。算法 A_{trans} 采用例 4.3 的方法，把 64 位的二进制位串变换成一个字符串。

算法 $A_{gen}(D_{salt}, D_{pw})$的工作过程如下：

（1）给口令 D_{pw} 撒盐：$D_{pw} = A_{salt}(D_{salt}, D_{pw})$；

（2）用撒盐结果做密钥：$K = D_{pw}$；

（3）用一个 64 位的全 0 的二进制位串构造一个数据块 D_p；

（4）设循环次数：$i = 0$；

（5）对数据块加密：$D_c = A_{crypt}(K, D_p)$；

（6）$D_p = D_c$，$i = i + 1$；

（7）如果 $i < 25$，则回到第（5）步；

（8）把数据块变换成字符串：$s = A_{trans}(D_c)$；

（9）返回 s。

第二部分：创建账户或修改用户口令时，账户信息数据库中的口令信息的维护。首先，确定一个随机数生成算法 A_{random}，它能生成随机数 D_{salt}，即

$$D_{salt} = A_{random}()$$

口令信息维护的工作过程如下：

（1）接收用户提供的口令 D_{pw}；

（2）生成一个盐值：$D_{salt} = A_{random}()$；

（3）生成口令信息：$s = A_{gen}(D_{salt}, D_{pw})$；

（4）把口令信息 s 和 D_{salt} 存入数据库的口令字段中。

第三部分：身份认证的工作过程如下：

（1）接收用户提供的账户名 D_{name} 和口令 D_{pw}；

（2）在账户信息数据库中检查 D_{name} 的合法性，如果合法，则找出其对应的 s 和 D_{salt}；

（3）生成临时口令信息：$s_r = A_{gen}(D_{salt}, D_{pw})$；

（4）如果 s_r 与 s 相等，则认证成功，否则认证失败。

例 4.6 是在例 4.2、例 4.3 和例 4.4 的基础上，结合撒盐措施，给出的一个综合的方案。传统 UNIX 操作系统中的用户口令信息管理和身份认证的方案与例 4.6 给出的方案是类似的。

传统 UNIX 操作系统使用 12 位的盐值，并且，在保存时，把它变换成由 2 个字符构成的字符串，作为前缀与口令相关的加密信息合在一起，存放在口令字段中。表 4-1 给出了口令、盐值和口令字段信息间对应关系的一个例子。

表 4-1　口令、盐值和口令字段的对应关系实例

口　令	盐　值	保存的口令字段信息
nutmeg	Mi	MiqkFWCm1fNJI
ellen1	ri	ri79KNd7V6.Sk
Sharon	./	./2aN7ysff3qM
norahs	am	amfIADT2iqjAf
norahs	7a	7azfT5tIdyh0I

从表 4-1 最下面的两行可看出，由于盐值的不同，同一个口令“norahs”可以对应到不同的相关加密信息。

3．口令信息与账户信息的分离

前面介绍了账户信息的管理方法。操作系统的账户信息存放在账户信息数据库中，UNIX 操作系统的/etc/passwd 文件是账户信息数据库的一种形式。传统的口令管理方法把口令管理信息存放在账户信息数据库中。以下是账户信息数据库中一个账户记录的例子。

例 4.7　以下是传统 UNIX 操作系统账户信息数据库中的一个账户记录信息样例：

wenchang:MiqkFWCm1fNJI:500:300:Wenchang Shi:/home/wenchang:/bin/bash

已知该系统采用的是撒盐式的口令管理，请简单说明该账户的口令字段信息的含义。

解答：账户记录的字段分隔符是冒号（:），第二个字段是口令字段，所以，口令字段信息是：

MiqkFWCm1fNJI

根据口令管理方法知：“Mi”是盐值变换后得到的字符串，含两个字符，“qkFWCm1fNJI”是加密结果变换后得到的字符串，含 11 个字符。其中，加密结果指的是以撒盐后的口令为密钥，对 64 位的 0 位串进行多次加密得到的结果。

通过例 4.7，我们进一步清楚地看到，口令字段信息既不是口令的明文，也不是口令的密文，只是与口令相关的信息而已。

不管口令字段信息具体内容是什么，它都是敏感信息，都是口令攻击的基础，最好不要暴露给无关的用户。由于每个用户都能够查看账户信息数据库，当口令字段信息存放在账户信息数据库中时，实际上，每个用户都能得到他相要的口令字段信息。为了消除这样的弊端，可以把口令字段信息从账户信息数据库中分离出去，建立专门的口令信息数据库，并且，只允许特权用户查看其中的内容。

例 4.8　为了提高口令信息的安全性，可以把口令信息从账户信息中分离出去，请给出一个在操作系统中存放口令信息的方案。

解答：仿照账户信息数据库的做法，用一个文本文件来表示口令信息数据库，库中为每个账户提供一个口令信息记录，每个记录用一行文本表示，每个记录文本中用冒号（:）表示字段分隔符，一个记录的字段构成如下：

$$T_{\text{name}}:T_{\text{pw}}:T_{\text{lstchg}}:T_{\text{min}}:T_{\text{max}}:T_{\text{warn}}:T_{\text{inact}}:T_{\text{expire}}:T_{\text{reserved}}$$

各字段的含义如下。

（1）T_{name}：账户名。

（2）T_{pw}：口令信息。

（3）T_{lstchg}：上次口令修改是在第 T_{lstchg} 天进行的（从 1970 年 1 月 1 日算起）。

（4）T_{min}：过了 T_{min} 天以后才能修改口令。

（5）T_{max}：过了 T_{max} 天以后必须修改口令。

（6）T_{warn}：口令过期前 T_{warn} 天提醒用户。

（7）T_{inact}：口令过期后 T_{inact} 天账户失效。

（8）T_{expire}：过了 T_{expire} 天后账户失效（从 1970 年 1 月 1 日算起）。

（9）T_{reserved}：保留字段。

除了账户名和口令信息外，其他信息主要是用来描述口令或账户的有效期限的。

现代的 UNIX 操作系统实现了口令信息与账户信息的分离，它们采用与例 4.7 类似的口令信息数据库，/etc/shadow 是这样的数据库的典型文件名。实现了口令信息与账户信息的分离后，账户信息数据库中的口令字段就不再存放口令信息，而只用一个 x 作为标示，图 4.1 所示的其实就是这种情形。图 4.5 是 UNIX 操作系统口令信息数据库文件/etc/shadow 的一个示例。

```
root:$1$jYTJgmNb$bJ5LQwc.91D4MMangK.Sm.:14028:0:99999:7:::
bin:*:14028:0:99999:7:::
daemon:*:14028:0:99999:7:::
rpc:!!:14028:0:99999:7:::
sshd:!!:14028:0:99999:7:::
wenchang:$1$k7TyrJaO$DS/P61XHIz1xWAqLcdnQz1:14091:0:99999:7:::
......
```

图 4.5　UNIX 操作系统口令信息数据库文件/etc/shadow 示例

图 4.5 中所示的口令字段信息与例 4.7 所示的传统口令字段信息格式有所不同，这是用功能更强的算法生成的。另外，口令字段中的“*”表示该账户不与任何口令相匹配，而“!!”表示该账户已被锁定。

4.3　面向网络的身份认证

在 4.2 节中，主要介绍了在单机系统上进行用户身份认证的基本方法。在网络环境中，进行身份认证所要考虑的问题比在单机环境中多很多。在单机环境身份认证的基础上，本节介绍在网络环境中进行身份认证需要解决的主要问题。

4.3.1　认证信息的网络化管理

在单机环境中，因为只有一台计算机可用，所以，计算机系统的所有服务和信息都部署在一台计算机上，用户登录计算机系统时，只需从自己操作的计算机上获取身份认证信息，便可完成身份认证工作。

在网络环境中，情况发生了很大变化，有大量的计算机在工作，用户从计算机系统中获取的很多服务都不是由他所操作的本地计算机提供的。在这样的背景下，用户希望不管操作哪台计算机，只要以相同的用户身份登录，都能获得所需要的服务。

基于本地主机的身份认证措施很难满足这样的要求。因为，如果把某用户的身份认证信息存放在某台本地的主机上，该用户就不太可能通过别的主机进行登录以获取所需的服务，如果在每台本地主机上都为该用户保存一份身份认证信息，则必然给身份认证信息的管理增加很大的难度。

把用户身份认证信息组织起来，集中存放到服务器中，借助服务器实现身份认证信息的共享，可以解决用户在网络中不同主机上的身份认证问题。

例 4.9　请参考本地主机上的身份认证方法的思路，给出一个在网络环境中进行用户身份认证的方法，要求使同一个合法用户可以在不同的主机上顺利进行身份认证。

解答：根据基于口令的身份认证思想，采用客户机/服务器模式，从身份认证信息的管理、客户机软件的功能和服务器软件的功能等方面给出解决问题的方法。

第一部分：身份认证信息的管理。设立一台身份认证信息数据库服务器，在服务器上，采用图 4.1 所示的账户信息格式、图 4.5 所示的口令信息格式、图 4.2 所示的用户组信息格式等，建立网络中合法用户的身份认证信息。

第二部分：客户机软件的功能。用户在客户机上登录时，客户机接收用户提供的账户名和口令信息后，把它们组织成身份认证请求，发送给服务器，由服务器为用户进行身份认证，如果服务器返回认证成功的结果，则登录成功，否则登录失败。

第三部分：服务器软件的功能。服务器接收到客户机发来的身份认证请求后，可以从其中得到相应的账户名和口令信息，同时，可以从服务器上的身份认证信息数据库中查找相应的账户名和口令信息，这时，可以采用类似例 4.6 的方法，判断账户名和口令的合法性，最后，把判断结果返回给客户机。

例 4.9 给出的方法是完全由服务器来负责认证工作的，客户机并不负责身份认证工作。这个方法从完全由客户机认证的一个极端，走向了完全由服务器认证的另一个极端。在这两个极端之间，有折中的方法，那就是由服务器和客户机分工进行认证。

例 4.10　请给出一个在网络环境中的客户机/服务器模式的用户身份认证方法，要求客户机和服务器都可以执行认证工作，允许根据用户来确定哪些认证由服务器执行，哪些认证由客户机执行。

解答：与例 4.9 类似，从身份认证信息的管理、客户机软件的功能和服务器软件的功能等方面给出解决问题的方法。

第一部分：身份认证信息的管理。

设立一台身份认证信息数据库服务器，在服务器上，采用图 4.1 所示的账户信息格式、图 4.5 所示的口令信息格式、图 4.2 所示的用户组信息格式等，建立网络中合法用户的身份认证信息。

同时，对客户机上的账户信息格式进行扩充，允许给账户记录信息添加一个“+”或“-”前缀，“+”前缀表示要由服务器对相应的账户执行认证，“-”前缀表示让服务器不要对相应的账户执行认证。

例如，下面的账户记录信息表示要由服务器对 weinan 账户执行认证：

root:x:0:0:root:/root:/bin/bash　　①

+weinan::501:301::: ②

下面的账户记录信息表示让服务器不要对 wenchang 账户执行认证：

root:x:0:0:root:/root:/bin/bash ③

-wenchang::500:300::: ④

+::999:999::: ⑤

我们规定，判断一个账户由服务器认证还是由客户机认证时，顺序扫描客户机上的账户记录信息，扫描到的第一个可匹配的账户记录信息决定判断的结果。在以上的账户记录信息中，第⑤行的信息表示所有的账户都由服务器认证，第④行的信息表示 wenchang 账户不要由服务器认证，由于扫描时先匹配的是第④行，所以，最终结果是 wenchang 账户不由服务器认证，而由客户机认证。

第二部分：客户机软件的功能。

用户在客户机上登录时，客户机接收用户提供的账户名和口令信息后，首先在客户机的账户记录信息中查询相应的账户，判断应该由服务器认证还是由客户机认证。

如果应该由服务器进行认证，则客户机把账户名和口令信息组织成身份认证请求，发送给服务器，由服务器为用户进行身份认证，如果服务器返回认证成功的结果，则登录成功，否则登录失败。

如果应该由客户机进行认证，则客户机从它保存的账户信息和口令信息数据库中查找相应的账户名和口令信息，采用类似于例 4.5 中的方法，判断账户名和口令的合法性，合法则登录成功，否则登录失败。

第三部分：服务器软件的功能。

服务器接收到客户机发来的身份认证请求后，可以从其中得到相应的账户名和口令信息，同时，可以从服务器上的身份认证信息数据库中查找相应的账户名和口令信息，这时，可以采用类似于例 4.5 中的方法，判断账户名和口令的合法性，最后，把判断结果返回给客户机。

美国 Sun 公司开发的 NIS（Network Information Service）系统实现了对例 4.10 中给出的方法的支持。

4.3.2　认证信息的加密传输

4.3.1 节讨论的方法达到了在网络环境中进行身份认证的目的，可是，其中存在一个明显的问题，那就是口令在网络中是以明文的形式传输的，这是一个严重的安全隐患。本节讨论解决这个问题的办法。

运用密码技术是实现信息在网络中安全传输的常用手段之一。密码体制包含秘密密钥密码体制和公开密钥密码体制。秘密密钥密码体制是单钥密码体制，使用一个秘密的密钥进行加密和解密。公开密钥密码体制是双钥密码体制，使用公钥和私钥两个密钥，

一个用于加密，另一个用于解密，公钥是对外公布的，私钥只有其拥有者才知道。

公开密钥密码算法的运行效率比较低，秘密密钥密码算法的运行效率比较高，把公开密钥密码算法和秘密密钥密码算法结合起来，可以设计出网络中的身份认证和信息加密传输方法。

例 4.11　设 Alice 和 Bob 是网络中需要相互通信的两个实体，请利用公开密钥密码算法和秘密密钥密码算法，给出一个 Alice 与 Bob 之间的身份认证和信息加密传输的方法。

解答：分别为 Alice 和 Bob 提供一对密钥($K_{\text{PUB-A}}$, $K_{\text{PRI-A}}$)和($K_{\text{PUB-B}}$, $K_{\text{PRI-B}}$)，其中，$K_{\text{PUB-A}}$ 和 $K_{\text{PRI-A}}$ 分别是 Alice 的公钥和私钥，$K_{\text{PUB-B}}$ 和 $K_{\text{PRI-B}}$ 分别是 Bob 的公钥和私钥。Alice 和 Bob 按照以下方式工作，以相互确认身份并进行信息的加密传输。

（1）Alice 生成一个会话密钥 K_{sess}，用 Bob 的公钥 $K_{\text{PUB-B}}$ 对它进行加密，结果可表示为

$$[K_{\text{sess}}]K_{\text{PUB-B}} \qquad ①$$

（2）Alice 用自己的私钥 $K_{\text{PRI-A}}$ 对结果①进行签名，结果可表示为

$$\{[K_{\text{sess}}]K_{\text{PUB-B}}\}K_{\text{PRI-A}} \qquad ②$$

（3）Alice 把结果②传送给 Bob。

（4）Bob 用 Alice 的公钥 $K_{\text{PUB-A}}$ 试图解开结果②。如果能解开，则得到结果①，同时证明结果②是由 Alice 发送的，由此确认 Alice 的身份。Bob 用自己的私钥 $K_{\text{PRI-B}}$ 对结果①进行解密，便得到 K_{sess}。

（5）Bob 用 Alice 的公钥 $K_{\text{PUB-A}}$ 对 K_{sess} 进行加密，结果可表示为

$$[K_{\text{sess}}]K_{\text{PUB-A}} \qquad ③$$

（6）Bob 用自己的私钥 $K_{\text{PRI-B}}$ 对结果③进行签名，结果可表示为

$$\{[K_{\text{sess}}]K_{\text{PUB-A}}\}K_{\text{PRI-B}} \qquad ④$$

（7）Bob 把结果④传送给 Alice。

（8）Alice 用 Bob 的公钥 $K_{\text{PUB-B}}$ 试图解开结果④。如果能解开，则得到结果③，同时证明结果④是由 Bob 发送的，由此确认 Bob 的身份。Alice 用自己的私钥 $K_{\text{PRI-A}}$ 对结果③进行解密，便得到 K_{sess}。如果这个 K_{sess} 就是 Alice 原来生成的那个 K_{sess}，则表示，经过协商，Alice 和 Bob 决定使用会话密钥 K_{sess}。

（9）Alice 和 Bob 开始进行信息传输，传输的信息采用 K_{sess} 进行加密和解密。

这个例子告诉我们，网络中的通信双方通过结合运用自己的私钥与对方的公钥，可以相互确认身份，同时，协商得到可用于通信信息的加密与解密的会话密钥。利用这个基础，可以给出在网络环境中保护用户口令安全的用户身份认证方法。

例 4.12　设有一个客户机/服务器模式的网络环境，请给出一个通过服务器认证客户

机上的用户的身份的方法，要求不能在网络中传输明文的或加密过的用户口令。

解答：给服务器分配一对密钥(K_{PUB-S}, K_{PRI-S})，给用户分配一对密钥(K_{PUB-U}, K_{PRI-U})，密钥都存放在服务器上，其中，用户的私钥 K_{PRI-U} 要加密之后再保存，即利用 DES 加密算法，以用户的口令 D_{pw} 为密钥对它进行加密，保存的加密结果可表示为

$$[K_{PRI-U}]\ D_{pw} \qquad ①$$

其他密钥直接以明文形式存放即可。

用户在客户机上登录时，客户机把用户的账户名传送给服务器，服务器把它自己的公钥 K_{PUB-S} 和用户的公钥 K_{PUB-U} 及结果①传送给客户机。客户机利用 DES 加密算法，以用户提供的口令为密钥，对结果①进行解密。如果用户提供的口令是正确的，则解密后得到用户的私钥 K_{PRI-U}。

此后，把用户的密钥对看做客户机的密钥对，把客户机和服务器分别看做 Alice 和 Bob，采取例 4.11 给出的方法进行认证，如果相互认证成功，则对用户的身份认证也就成功了。

这个例子给出的方法把用户口令与用户私钥联系到了一起，通过解密和通信双方的相互确认达到了对用户进行身份认证的目的，同时为下一步的加密通信做好了准备。在整个认证过程中，并没有在网络中传输用户的口令。

例 4.13　设有一个客户机/服务器模式的网络环境，请给出在客户机上登录的用户身份认证方法，要求客户机和服务器都可以执行认证工作，允许根据用户来确定哪些认证由服务器执行，哪些认证由客户机执行，而且，不能在网络中传输明文的或加密过的用户口令。

解答：按照例 4.10 的方法管理用户身份认证信息，同时，在服务器上增加服务器和用户的密钥对信息，按照例 4.12 的方法保存。

当用户在客户机上登录时，客户机按照例 4.10 的方法判断该由客户机还是服务器执行身份认证。如果需要由服务器执行认证，则按照例 4.12 的方法进行，如果需要由客户机执行认证，则按照例 4.10 的方法进行。

美国 Sun 公司开发的 Secure RPC 实现了对例 4.12 中给出的方法的支持，而 Sun 公司开发的以 Secure RPC 为基础的 NIS+系统实现了对例 4.13 中给出的方法的支持。

4.3.3　面向服务的再度认证

去北京的颐和园游玩过的朋友都知道，进颐和园是需要购门票的。有了门票，你已经可以尽情欣赏这座中国皇家园林的恢弘气势。不过，如果还想在昆明湖上荡舟，或者到佛香阁上远眺，那么，还需要另外专门购票，因为，这些属于特定的服务项目。

社会生活中的这种模式可以类比到网络环境下的信息系统中。在本节的讨论中，我

们借用颐和园的门票和游船票的概念，在认证系统中引入两张票，为叙述方便，不妨分别称为通行证和服务卡。成功登录进入系统的用户得到通行证，凭借通行证可以申请服务卡，有了服务卡就可以获得特定的服务。

例 4.14　设在一个网络环境中，用户必须拥有服务认可才能获得特定的服务，请给出用户身份认证和服务认证的一种方法，要求不能在网络中传输明文的或加密过的用户口令，而且，不能使用公开密钥密码算法。

解答：采用秘密密钥密码算法进行加/解密。建立一个身份认证服务器 S_{auth} 和一个服务审批服务器 S_{grant}，设应用服务器为 S_{serv}。在服务器 S_{auth} 上保存用户的账户和口令及服务器 S_{auth} 的密钥，在服务器 S_{grant} 上保存服务器 S_{serv} 提供的服务的密钥。认证的主要步骤如下。

（1）用户在客户机上登录时，客户机把用户的账户名传送给服务器 S_{auth}，请求进行身份认证。

（2）服务器 S_{auth} 验证账户名的合法性，如果合法，则生成会话密钥 $K_{\text{sess-1}}$ 和通行证 T_{grant}，其中，$K_{\text{sess-1}}$ 也成为 T_{grant} 中的内容之一。用服务器 S_{grant} 的密钥 K_{grant} 加密 T_{grant}，结果可表示为

$$[T_{\text{grant}}]K_{\text{grant}} \qquad ①$$

把 $K_{\text{sess-1}}$ 和结果①组合在一起，并以用户口令 D_{pw} 为密钥进行加密，结果可表示为

$$[K_{\text{sess-1}}, [T_{\text{grant}}]K_{\text{grant}}]\, D_{\text{pw}} \qquad ②$$

再把结果②传送给客户机。

（3）客户机以用户提供的口令为密钥对结果②进行解密，如果解密成功，则表明用户提供了正确的口令，登录成功，并得到会话密钥 $K_{\text{sess-1}}$ 和结果①。

用户要求获得某项服务时，客户机为它生成申请服务的请求 Q_{grant}，并用会话密钥 $K_{\text{sess-1}}$ 对它进行加密，结果可表示为

$$[Q_{\text{grant}}]K_{\text{sess-1}} \qquad ③$$

再把结果③和结果①传送给服务器 S_{grant}。

（4）服务器 S_{grant} 用自己的密钥 K_{grant} 对结果①进行解密，得到 T_{grant}。从 T_{grant} 中可以得到 $K_{\text{sess-1}}$。用 $K_{\text{sess-1}}$ 对结果③进行解密得到 Q_{grant}。

根据 Q_{grant} 和 T_{grant} 分析申请服务的请求并验证其合法性，如果合法，则生成会话密钥 $K_{\text{sess-2}}$ 和服务卡 T_{serv}，其中，$K_{\text{sess-2}}$ 也成为 T_{serv} 中的内容之一。用服务器 S_{serv} 的密钥 K_{serv} 加密 T_{serv}，结果可表示为

$$[T_{\text{serv}}]K_{\text{serv}} \qquad ④$$

把 $K_{\text{sess-2}}$ 和结果④组合在一起，并用会话密钥 $K_{\text{sess-1}}$ 进行加密，结果可表示为

$$[K_{\text{sess-2}}, [T_{\text{serv}}]K_{\text{serv}}]\ K_{\text{sess-1}} \qquad ⑤$$

再把结果⑤传送给客户机。

（5）客户机用会话密钥 $K_{sess\text{-}1}$ 对结果⑤进行解密，得到会话密钥 $K_{sess\text{-}2}$ 和结果④。客户机生成启动服务的请求 Q_{serv}，并用 $K_{sess\text{-}2}$ 对它进行加密，结果可表示为

$$[Q_{serv}]\ K_{sess\text{-}2} \qquad ⑥$$

再把结果⑥和结果④传送给服务器 S_{serv}。

（6）服务器 S_{serv} 用自己的密钥 K_{serv} 对结果④进行解密，得到 T_{serv}。从 T_{serv} 中可以得到 $K_{sess\text{-}2}$。用 $K_{sess\text{-}2}$ 对结果⑥进行解密得到 Q_{serv}。根据 Q_{serv} 和 T_{serv} 分析启动服务的请求，并验证其合法性，如果合法，则启动相应的服务。

这个例子展示了进行用户身份认证和服务请求认证的思想。美国麻省理工学院（MIT）开发的 Kerberos 认证系统实现了对例 4.14 中给出的方法的支持。

4.4　基于PAM的统一认证框架

经过前面的学习，我们已经能够看出，系统中的身份认证方法是多样的，如传统 UNIX 认证方法、NIS+认证方法、Kerberos 认证方法等。同时，也不难想象，系统中要求认证用户身份的服务程序也不少，如 login、ftp、telnet、ssh 等。

在系统设计上，如果让每种服务程序各自拥有自己的认证机制，那么，系统中必然存在大量的认证机制和大量的用户身份数据信息，而这些机制和信息中的很多内容可能是重复的或相似的，这使系统的管理和维护变得非常困难。

为了使不同的服务程序能够共享相同的认证机制和信息，为了使同一个服务程序能够灵活地选择不同的认证方法，从而简化系统的管理和维护工作，提高系统的灵活性，我们需要一个统一的身份认证框架。

PAM（Pluggable Authentication Modules）就是一个统一的身份认证框架，起初，它是由美国 Sun 公司为 Solaris 操作系统开发的，后来，很多操作系统都实现了对它的支持。

PAM 的基本思想是实现服务程序与认证机制的分离，通过一个插拔式的接口，让服务程序插接到接口的一端，让认证机制插接到接口的另一端，从而实现服务程序与认证机制的随意组合，如图 4.6 所示。

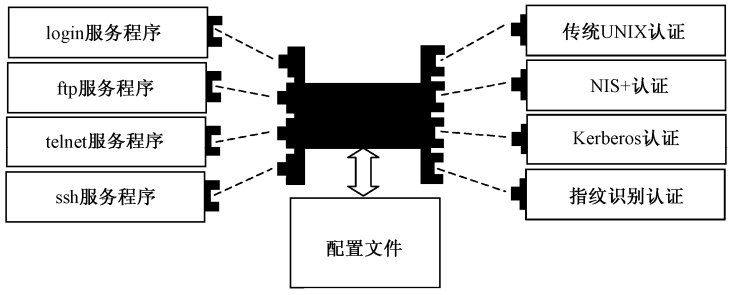


图 4.6　PAM 认证框架的基本思想

PAM 框架定义了一个应用编程接口（API，Application Programming Interface）。一个 PAM 系统主要由 API、动态装载的共享库和配置文件构成。在 PAM 框架中，每一种身份认证机制设计为一个 PAM 模块，实现为一个动态装载的共享库。PAM 模块遵循 PAM 的 API 规范。需要实施身份认证过程的服务程序按照 PAM 的 API 规范调用 PAM 系统中的身份认证功能。图 4.7 描述了 PAM 框架的基本结构和工作原理。

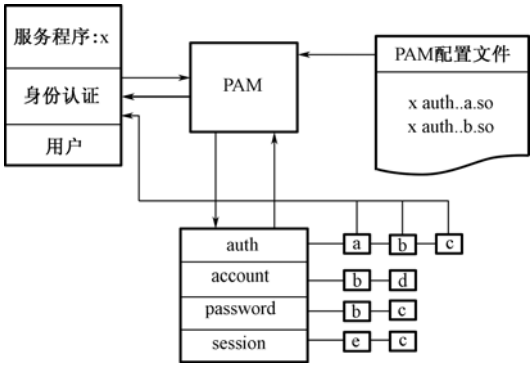


图 4.7　PAM 框架的基本结构和工作原理

如图 4.7 所示，x 是一个需要实施身份认证过程的服务程序。用户启动程序 x 时，程序首先进入身份认证状态，它把身份认证的任务交给 PAM 系统去完成。PAM 系统根据配置文件，确定需要为程序 x 提供的身份认证支持服务。PAM 提供四种类型的支持服务，分别由 auth、account、password 和 session 标识，它们分别指定身份认证、账户管理、口令管理和会话管理时需要调用的共享库。图 4.7 中，a、b、c 等表示的就是共享库。

例 4.15　设以下是在某 UNIX 操作系统的 PAM 配置文件/etc/pam.conf 中为 OpenSSH 服务程序定义的配置信息：

```
sshd auth        required    /lib/security/pam_env.so
sshd auth        sufficient  /lib/security/pam_unix.so likeauth nullok
sshd auth        required    /lib/security/pam_deny.so
sshd account     required    /lib/security/pam_unix.so
sshd password    required    /lib/security/pam_cracklib.so retry=3
sshd password    sufficient  /lib/security/pam_unix.so nullok use_authtok md5 shadow
sshd password    required    /lib/security/pam_deny.so
sshd session     required    /lib/security/pam_limits.so
sshd session     required    /lib/security/pam_unix.so
```

请简要说明配置信息的组成结构，以及 PAM 系统为 OpenSSH 服务程序完成身份认证相关工作的基本方法。

解答：配置信息的每一行定义一条执行模块的命令，每行信息共分为四列，第一列

是服务程序的名称，如 sshd；第二列是 PAM 提供的服务的类型，如 auth；第三列是执行命令时的控制标记，如 required；第四列是要调用的共享库名，如 pam_env.so，必要时，还有调用共享库时使用的参数。

PAM 提供 requisite、required、sufficient 和 optional 四种控制标记。标记为 requisite 的命令执行失败时，PAM 立刻返回并报告失败；标记为 required 的命令执行失败时，PAM 继续执行后续命令，但最终报告失败；标记为 sufficient 的命令执行成功时，如果之前没有出现过失败的命令，则 PAM 立刻返回并报告成功；标记为 optional 的命令的执行结果不影响其他命令的执行，也不影响 PAM 报告的结果。

当 OpenSSH 服务程序要进行身份认证时，PAM 为它执行三条命令，第一条命令设置环境变量，第二条命令进行传统 UNIX 式的身份认证，如果该命令成功，则用户在 OpenSSH 上的身份认证成功，否则执行第三条命令，拒绝进入系统，认证失败。

当 OpenSSH 服务程序要进行账户管理、口令管理或会话管理时，PAM 分别为它执行一条、三条、两条命令。

例 4.16　在例 4.15 中，OpenSSH 服务的 PAM 配置是在配置文件/etc/pam.conf 中定义的，我们也可以用配置文件/etc/pam.d/sshd 来定义它的 PAM 配置，请给出用后一个文件来定义的配置信息。

解答：在/etc/pam.d/sshd 中定义的配置信息与在/etc/pam.conf 中定义的配置信息基本相同，只是去掉了第一列信息，即去掉了每行信息中的服务程序名 sshd。

4.5　基于权限位的访问控制

操作系统中最朴素的自主访问控制思想当属控制用户对文件的访问。访问控制模型的三个基本要素是主体、客体和访问方式。无疑，在操作系统的访问控制体系中，用户是最直观的主体，文件是最直观的客体，最直观的访问方式则是用户执行的对文件的操作。

4.5.1　访问权限的定义与表示

用户对文件的操作可以归纳为三种形式，即查看文件中的信息、改动文件中的信息、或运行文件所表示的程序。与之相对应，可以为用户对文件的操作定义读、写和执行三种方式，可以分别用 r、w 和 x 三个字符来表示。也就是说，一个用户可以对一个文件进行 r、w、x 三种操作。

用户从操作系统中获得的以某种方式对文件进行操作的许可，就是用户对文件进行访问的权限，因此可以说，用户可以拥有对文件进行 r、w、x 三种权限。拥有 r、w、x 权限分别表示操作系统允许用户对文件进行读、写、执行操作。

在人机交互中，对于用户来说，用 r、w、x 等字符来表示访问权限非常直观，而对

于操作系统来说，用二进制位来表示访问权限则更高效。

例 4.17　设在操作系统中，针对用户对文件的操作，定义了读、写、执行三种访问权限，请给出一个用二进制位来表示用户拥有的对文件的访问权限的方法。

解答：用一个由 3 个二进制位组成的位串来表示一个用户拥有的对一个文件的所有访问权限，每种访问权限由 1 个二进制位来表示，由左至右，位串中的各个二进制位分别对应读、写、执行权限。二进制位的值与访问权限的关系：1 表示拥有权限，0 表示未拥有权限，位串的赋值与用户拥有的访问权限的结果如下。

000：不拥有任何权限；

001：拥有执行权限，不拥有读、写权限；

010：拥有写权限，不拥有读、执行权限；

011：拥有写和执行权限，不拥有读权限；

100：拥有读权限，不拥有写、执行权限；

101：拥有读和执行权限，不拥有写权限；

110：拥有读和写权限，不拥有执行权限；

111：拥有读、写和执行权限。

例 4.18　设在操作系统内部，用例 4.17 给出的二进制位串表示用户拥有的访问权限，请给出一个把二进制位串映射为字符串的方法。

解答：用一个由 3 个字符组成的字符串表示一个用户拥有的对一个文件的所有访问权限，字符串与二进制位串的对应关系，从左至右，一个字符对应一个二进制位，二进制位为 0 时，对应的字符取减号（-）；二进制位为 1 时，对应的字符如下。

第一个二进制位为 1：第一个字符取 r；

第二个二进制位为 1：第二个字符取 w；

第三个二进制位为 1：第三个字符取 x。

例如，二进制位串 000 和 111 分别对应字符串“---”和“rwx”。

一个 3 位的二进制数与一个 1 位的八进制数对应，因此，可以用一个 1 位的八进制数表示一个用户拥有的对一个文件的所有访问权限。

例 4.19　设用户 U_1 对文件 F_1、F_2、F_3 拥有的权限的八进制值分别为 1、2、4，请问，该用户对这三个文件分别可以进行什么操作？

解答：八进制值 1、2、4 对应的二进制值分别为 001、010、100，所以，用户 U_1 对文件 F_1、F_2、F_3 分别可以进行执行、写、读操作。

例 4.20　设用户 U 对文件 F 拥有“r-x”权限，请问，该用户对该文件可以进行什么操作？该用户拥有的权限的二进制和八进制表示分别是什么？

解答：用户 U 对文件 F 可以进行读和执行操作，用户 U 对文件 F 拥有的权限的二进制和八进制表示分别为 101 和 5。

4.5.2　用户的划分与访问控制

操作系统可以为系统中的每个文件定义一个属主（Owner）。通常，如果一个用户创建了一个文件，那么，该用户就是该文件的属主。当然，如果用户 U_1 是文件 F_1 的属主，他可以把 F_1 赠送给用户 U_2，这样，用户 U_2 就成了文件 F_1 的属主，而用户 U_1 就不再是文件 F_1 的属主。一个文件只有一个属主。

操作系统一般都支持对用户进行分组管理，系统中的一个或多个用户可以组成一个用户组。当一个用户参与组成一个用户组时，该用户就是该用户组的成员，该用户组可以称为该用户的属组。

出于方便地进行访问控制的需要，针对一个给定的文件，可以简单地把系统中的用户划分成三个用户域，系统中用户域的划分如图 4.8 所示。

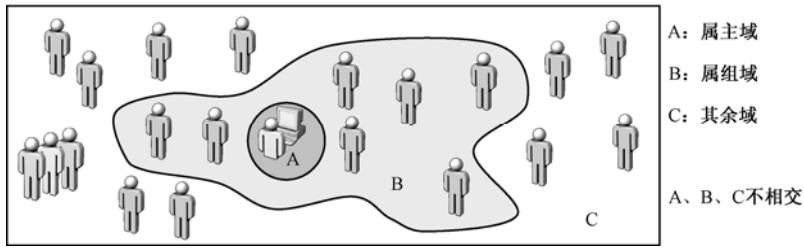


图 4.8　系统中用户域的划分

其中，第一个域由文件的属主构成，称为属主域，只包含一个用户，如图 4.8 中的 A 区。第二个域由文件的属主的属组中的用户构成，称为属组域，可包含一个或多个用户，如图 4.8 中的 B 区。第三个域由系统中属主和属组以外的所有用户构成，称为其余域，包含多个用户，如图 4.8 中的 C 区。A、B、C 三个区互不相交。

一个用户很有可能参加多个用户组，如果是这种情形，则可以确定其中一个用户组为主用户组，属组域由主用户组定义。

4.5.1 节定义了一个用户对一个文件的访问权限，实际上，该定义并不局限于一个用户，可以用于一类用户，表示一类用户对一个文件的访问权限。

把每个用户域中的用户看做一类用户，则系统中的用户便分成了三类，分别是属主类、属组类和其余类。我们可以同时定义三类用户对一个文件的访问权限。

一类用户对一个文件的访问权限可以由 3 个二进制位表示，因此，三类用户对一个文件的访问权限可以由 9 个二进制位表示。

例 4.21　设操作系统中的用户可以划分为属主、属组和其余三类，请给出一个用二进制位表示用户对文件的访问权限的方法，要求对任意一个给定的文件，可以确定每类

用户对它的访问权限。

解答：用一个由 9 个二进制位组成的位串来表示用户对一个文件的访问权限，其中，左边 3 个二进制位、中间 3 个二进制位、右边 3 个二进制位分别表示属主类、属组类、其余类用户对文件的访问权限。

例如，如果文件 F 的二进制权限位串是 111101001，则三类用户的访问权限分别是 111、101、001。即，属主类用户对文件 F 拥有读、写和执行权限；属组类用户对文件 F 拥有读和执行权限，其余类用户对文件拥有执行权限。

例 4.22　设操作系统中，用户对文件 F 的访问权限的二进制表示为 111101001，请给出用户对该文件的访问权限的八进制和字符串形式的表示。

解答：三类用户对文件 F 的访问权限的二进制表示分别为 111、101、001，相应的八进制表示分别是 7、5、1，相应的字符串表示分别是 rwx、r-x、--x，所以，用户对文件 F 的访问权限的八进制和字符串形式的表示分别是 751 和 rwxr-x--x。

根据以上“属主/属组/其余”式的用户分类方法，对于系统中的任何一个用户，都必然有相应的用户类型与其对应。当一个用户试图访问一个文件时，只要我们为该文件定义了三类用户对它的访问权限，就一定能找到与该用户匹配的访问权限，从而控制该用户对该文件的访问。

所以，通过为操作系统中的每一个文件定义“属主/属组/其余”式的访问权限，可以实现操作系统中所有用户对所有文件的访问控制。操作系统可以为每个新创建的文件定义默认的访问权限。在自主访问控制中，文件的属主可以修改文件的访问权限。

4.5.3　访问控制算法

4.5.2 节介绍了“属主/属组/其余”式的访问控制思想，其中的属组的定义是文件的属主的属组，其实，也可以称为文件的属组。该访问控制思想通过 9 个二进制权限位来表示用户对文件的访问权限，因而，也可以称为基于权限位的访问控制。下面讨论根据这种访问控制思想进行访问控制的实施算法。

例 4.23　设某操作系统采取“属主/属组/其余”式的访问控制思想对用户访问文件的行为进行控制，请给出一个进行访问控制判定的算法。

解答：设用户 U 请求对文件 F 进行 a 操作，其中 a 是 r、w 或 x，文件 F 的属主和属组分别为 U_o 和 G_o，按照以下步骤进行访问控制判定。

（1）当 U 等于 U_o 时，如果文件 F 的 9 位权限位组的属主位组中与 a 对应的位为 1，则允许 U 对 F 进行 a 操作，否则，不允许 U 对 F 进行 a 操作，判定结束。（属主位组由 9 位权限位组的左边 3 位组成。）

（2）当 G_o 是 U 的属组时，如果文件 F 的 9 位权限位组的属组位组中与 a 对应的位

为 1，则允许 U 对 F 进行 a 操作，否则，不允许 U 对 F 进行 a 操作，判定结束。（属组位组由 9 位权限位组的中间 3 位组成。）

（3）如果文件 F 的 9 位权限位组的其余位组中与 a 对应的位为 1，则允许 U 对 F 进行 a 操作，否则，不允许 U 对 F 进行 a 操作。（其余位组由 9 位权限位组的右边 3 位组成。）

这个算法首先确定用户是“属主”、“属组”和“其余”中的哪类用户，然后根据为该类用户分配的权限进行判定。

例 4.24　设在某 UNIX 操作系统中，部分用户组的配置信息如下：

sisefellow:x:300:wenchang,binliang,zhiyong,zhaohui

siselab_ms:x:301:weinan,hanchao,kankan,xiaoli,liuxing

系统中部分文件的权限配置信息如下：

rw-r-x--x　wenchang　sisefellow　..... file1

r---w---x　weinan　　siselab_ms file2

其中，配置信息的第一列是权限，第二列是属主名，第三列是属组名，最后一列是文件名。请问，用户 wenchang、binliang 和 weinan 可以对文件 file1 进行什么操作？用户 hanchao 可以对文件 file2 进行什么操作？

解答：用户 wenchang 是文件 file1 的属主，可以对文件 file1 进行读和写操作。

用户 binliang 是文件 file1 的属组的成员，所以，该用户拥有分配给文件 file1 的属组的权限，即用户 binliang 可以对文件 file1 进行读和执行操作。

用户 weinan 既不是文件 file1 的属主，也不是文件 file1 的属组的成员，所以，该用户拥有分配给其余用户的权限，即用户 weinan 可以对文件 file1 进行执行操作。

用户 hanchao 是文件 file2 的属组的成员，所以该用户拥有分配给文件 file2 的属组的权限，即用户 hanchao 可以对文件 file2 进行写操作。

例 4.25　设在某 UNIX 操作系统中，自主访问控制机制的部分相关文件配置信息如下：

r--r--r--　weinan　　siselab_ms file3

请问，用户 weinan 是否有可能对文件 file3 进行写操作？

解答：从文件 file3 的权限配置信息来看，用户 weinan 不拥有对文件 file3 的写权限，该用户不能对文件 file3 进行写操作，但是，该用户是文件 file3 的属主，根据自主访问控制性质，该用户可以修改文件 file3 的权限配置，例如，修改为如下形式：

rw-r--r--　weinan　　siselab_ms file3

这时，用户 weinan 便拥有了对文件 file3 的写权限，所以，用户 weinan 有可能对文件 file3 进行写操作。

传统 UNIX 操作系统的自主访问控制机制实现了对“属主/属组/其余”式的访问控

制思想的支持。

4.6　进程的有效身份与权限

前面曾经说过，用户是最直观的主体。但是，在操作系统中，进程才是真正活动的主体，进程在系统中代表用户进行工作，用户对系统的操作是由进程代其实施的。

进程是程序的执行过程，而程序是由文件表示的，所以，进程与文件有密切的关系。进程又是代表用户进行工作的，所以，进程与用户也有很大的关系。首先，认识一下进程、文件和用户之间的关系。

4.6.1　进程与文件和用户的关系

用户通过操作系统进行工作时，会启动相应的进程，该进程执行操作系统中的相应可执行文件，为用户完成工作任务。可执行文件以程序映像的形式装入到进程之中，成为进程的主体成分，构成进程的神经系统，指挥进程一步一步地开展工作。用户与进程和文件的关系如图 4.9 所示，描绘了用户启动进程执行程序文件的基本思想。

图 4.9 中，用户 U_p 启动了进程 P，进程 P 运行可执行文件 F 中的程序，文件 F 的属主是用户 U_f。进程 P 在操作系统中代表用户 U_p 进行工作。例如，用户 U_p 查看文件 filex，实际上就是进程 P 读文件 filex 的内容并把它显示出来。

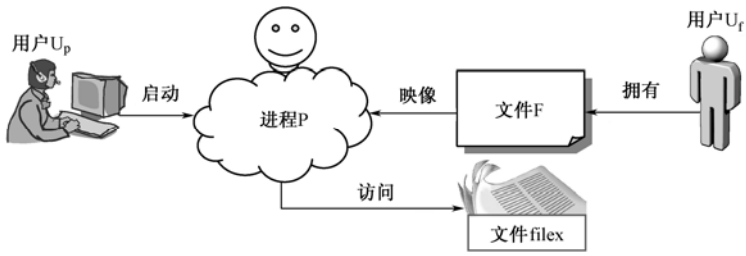


图 4.9　用户与进程和文件的关系

用户 U_p 查看文件 filex，需要拥有对文件 filex 的读权限，同样，进程 P 读文件 filex，也必须要拥有对文件 filex 的读权限。当然，对于写或执行操作，也是一样的道理。如何确定进程访问文件的权限呢？因为进程是由用户创建的，因此，可以借助用户的访问权限来确定进程的访问权限。

例 4.26　设在操作系统中，进程必须拥有对文件的访问权限才能对文件进行相应的访问，请给出一个确定进程对文件的访问权限的方法。

解答：设进程 P 是由用户 U_p 启动的，对任意的文件 filex，使进程 P 对文件 filex 的访问权限等于用户 U_p 对文件 filex 的访问权限。

这个例子给出的方法把启动进程的用户对文件的访问权限作为进程对文件的访问权限，这是自然的，因为进程是用户的化身，是代表用户工作的。

图 4.9 中涉及两个用户，即除了进程 P 的启动者 U_p 外，还涉及进程 P 所执行的文件 F 的属主 U_f。是否可以把用户 U_f 对文件的访问权限作为进程 P 对文件的访问权限呢？当然也是可以的。

例 4.27　设在操作系统中，进程必须拥有对文件的访问权限才能对文件进行相应的访问，请给出一个确定进程对文件的访问权限的方法。

解答：设进程 P 运行的程序是文件 F，文件 F 的属主是用户 U_f，对任意的文件 filex，使进程 P 对文件 filex 的访问权限等于用户 U_f 对文件 filex 的访问权限。

显然，例 4.26 给出的方法让进程 P 以用户 U_p 的身份去访问文件 F，而例 4.27 给出的方法让进程 P 以用户 U_f 的身份去访问文件 F。

用户 U_p 启动的进程 P 运行程序文件 F，实际上就是用户 U_p 对文件 F 进行执行操作，所以，要求用户 U_p 拥有对文件 F 的执行权限。

4.6.2　进程的用户属性

通过 4.6.1 节的讨论可知，可以借鉴用户的访问控制方法设计进程的访问控制方法。用户的访问控制根据用户属性和文件的访问属性进行判定，用到的用户属性是用户标识和用户组标识，用到的文件属性是文件属主、文件属组和访问权限位串。作为一种借鉴，可以为进程设立用户标识和用户组标识属性，作为访问判定的依据。

例 4.28　设在操作系统中，进程中有用户标识和用户组标识属性，文件中有属主、属组和访问权限位串属性，请给出进程访问控制的一种方法，要求根据这些进程属性和文件属性进行访问判定。

解答：设任意进程 P 请求对任意文件 F 进行访问，进程 P 的用户标识和用户组标识分别为 I_{up} 和 I_{gp}，文件 F 的属主、属组和访问权限位串分别为 I_{uf}、I_{gf} 和 $S_1S_2S_3$，其中，S_1、S_2、S_3 分别表示 9 位的访问权限位串中左、中、右三个 3 位的子位串，按照以下步骤进行判定。

（1）当 I_{up} 等于 I_{uf} 时，检查 S_1 中是否有相应权限，如果有，则允许访问，否则不允许访问，结束判定。

（2）当 I_{gp} 等于 I_{gf} 时，检查 S_2 中是否有相应权限，如果有，则允许访问，否则不允许访问，结束判定。

（3）检查 S_3 中是否有相应权限，如果有，则允许访问，否则不允许访问。

需要考虑如何确定进程中的用户标识的问题。在图 4.9 所示的示例中，进程 P 是由用户 U_p 创建的，但例 4.27 中给出的方法根据用户 U_f 的标识进行访问判定，而不是根据

用户 U_p 的标识进行访问判定。有必要记住两类用户，即创建进程的用户和借以进行访问判定的用户。

例 4.29　请给出一种在进程中设立用户属性的方法，要求能够反映创建进程的用户和借以进行访问控制的用户。

解答：在进程中设立两套用户属性，一套用于记住创建进程的用户，一套用于进行访问判定，每套用户属性都包含一个用户标识和一个用户组标识。

用于记住创建进程的用户的属性称为真实用户属性，相应标识分别称为真实用户标识（简记为 RUID）和真实用户组标识（简记为 RGID）。用于进行访问判定的属性称为有效用户属性，相应标识分别称为有效用户标识（简记为 EUID）和有效用户组标识（简记为 EGID）。

进行进程访问控制时，使用有效用户标识（EUID）和有效用户组标识（EGID），按照例 4.28 给出的方法进行访问判定。

应用例 4.29 给出的问题描述方法，则例 4.26 中采取的策略是使 EUID 和 EGID 与用户 U_p 及其属组对应，例 4.27 中采取的策略是使 EUID 和 EGID 与用户 U_f 及其属组对应，两个例子中的 RUID 和 RGID 都与用户 U_p 及其属组对应。

4.6.3　进程有效用户属性的确定

首先看一下进程在它的生命周期中的成长变化情况。进程是运行中的程序，进程所运行的程序决定了进程的本质，只要更换进程所运行的程序，不用创建新的进程，就能改变进程的本质，使进程执行新的任务。

例 4.30　请给出操作系统进程控制的一种方法，要求能够使任意一个现有进程在不结束生命的前提下执行新进程。

解答：在操作系统中设计一个系统调用，它的功能就是把调用它的进程所运行的程序替换成一个新的程序，不妨把它表示为 exec()，调用形式为

exec(“progf”)

其中，progf 是一个可执行程序文件名，该系统调用把调用它的进程所运行的程序替换成 progf，这相当于把正在运行的程序彻底清除掉，然后用程序 progf 来代替它。

例 4.31　设操作系统提供更新进程的 exec()系统调用，已知三个可执行程序的伪代码如下：

```
progf1：
        printf( “China” );
        exec( “progf2” );
        printf( “America” );
```

```
            return;
progf2：
            printf( “England” );
            exec( “progf3” );
            printf( “Canada” );
            return;
progf3：
            printf( “Australia” );
            return;
```

某用户执行程序 progf1 启动了进程 proc1，请问进程 proc1 在运行过程中显示什么信息？请按顺序把它们列出来。

解答：进程显示的信息依次是：China — England — Australia。

可以在创建进程时和更新进程的程序映像时确定进程的用户属性。

例 4.32　设操作系统中的进程可以通过系统调用 exec()更新程序映像，请给出一个在进程的整个生命周期中确定进程的用户属性的方法。

解答：设用户 U 创建进程 P，进程 P 的 RUID、RGID、EUID 和 EGID 分别为 I_{up}、I_{gp}、I_{ue} 和 I_{ge}，F 是一个任意的可执行程序文件，文件 F 的属主和属组的标识分别为 I_{uf} 和 I_{gf}，确定进程 P 的用户属性的方法如下。

（1）用户 U 创建进程 P 时，设

$$I_{up}=I_{ue}=\text{用户 U 的标识}$$

$$I_{gp}=I_{ge}=\text{用户 U 的属组的标识}$$

（2）进程 P 调用 exec(“F”)把程序映像替换为 F 时，如果 I_{uf} 的条件允许，则设

$$I_{ue}=I_{uf}$$

如果 I_{gf} 的条件允许，则设

$$I_{ge}=I_{gf}$$

例 4.32 中的方法涉及 I_{uf} 和 I_{gf} 的条件问题，这可以通过扩充文件的二进制访问权限位串来解决。

例 4.33　请给出一种扩充文件的二进制访问权限位串的方法，以便在进程更新程序映像时能够确定是否可以修改进程的 EUID 和 EGID。

解答：设 P 为任意进程，对于任意的文件 F，现有的 9 位二进制访问权限位串可以表示为

$$r_o w_o x_o r_g w_g x_g r_a w_a x_a$$

在该位串的左边增加 3 个二进制位，扩充为以下形式的 12 位的位串：

$$u_t g_t s_t r_o w_o x_o r_g w_g x_g r_a w_a x_a$$

其中，u_t 和 g_t 用于控制对进程的 EUID 和 GUID 的更新，可分别称为 SETUID 控制位和 SETGID 控制位，s_t 暂时不用。

当进程 P 调用 exec(“F”)把程序映像替换为 F 时，控制方法定义如下。

（1）u_t=1：允许进程 P 的 EUID 值取文件 F 的属主标识；

（2）u_t=0：不允许进程 P 的 EUID 值取文件 F 的属主标识；

（3）g_t=1：允许进程 P 的 GUID 值取文件 F 的属组标识；

（4）g_t=0：不允许进程 *P* 的 GUID 值取文件 *F* 的属组标识。

这个例子扩充了文件的访问权限属性结构，把 9 位的权限位串扩展为 12 位的权限位串，增设了 SETUID 控制位和 SETGID 控制位，显然，这些控制位仅对可执行文件有意义。

例 4.34　请给出一个把 12 位的二进制权限格式转换成字符串权限格式的方法。

解答：12 位二进制权限格式可以表示为

$$u_t g_t s_t r_o w_o x_o r_g w_g x_g r_a w_a x_a \qquad (1)$$

例 4.18 给出了把 $r_o w_o x_o$、$r_g w_g x_g$ 和 $r_a w_a x_a$ 转换成字符串格式的方法，设转换得到的结果分别表示为 $R_o W_o X_o$、$R_g W_g X_g$ 和 $R_a W_a X_a$，则

$$r_o w_o x_o r_g w_g x_g r_a w_a x_a \qquad (2)$$

转换后得到：

$$R_o W_o X_o R_g W_g X_g R_a W_a X_a \qquad (3)$$

以（3）为基础，当（1）中的 u_t 是 1 时，把（3）中的 X_o 设为 s；当（1）中的 g_t 是 1 时，把（3）中的 X_g 设为 s，这样对（3）进行修改后得到的结果就是与二进制格式（1）对应的字符串格式。例如：

100101001001 转换后的结果是 r-s--x--x

010101001001 转换后的结果是 r-x--s--x

这个例子在 9 位二进制权限位串对应的 9 字符权限格式的基础上，提供了一种表示 SETUID 和 SETGID 控制位的简捷方法，依然用 9 字符权限格式，能表示 12 位二进制权限位串。

至此，借助文件的访问权限属性和更新进程程序映像的 exec()系统调用，便能确定进程的有效用户属性，而根据进程的有效用户属性和文件的访问权限属性，就可以实现进程访问文件时的访问控制。

例 4.35　设在某 UNIX 操作系统中，部分用户组的配置信息如下：

siselab_ms:x:301:weinan,hanchao,kankan,xiaoli,liuxing

系统中部分文件的权限配置信息如下：

--x--x--x　wenchang　　sisefellow progf1

--x--s--x　hanchao　　siselab_ms progf2

```
--s--x--x    weinan        siselab_ms ..... progf3
rw-r-----    weinan        siselab_ms ..... filex
```

程序 progf1、progf2 和 progf3 的伪代码如例 4-6-6 所示。用户 wenchang 执行程序 progf1 启动了进程 P，请问：

（1）进程 P 在显示 China 时，对文件 filex 拥有什么访问权限？

（2）进程 P 在显示 England 时，对文件 filex 拥有什么访问权限？

（3）进程 P 在显示 Australia 时，对文件 filex 拥有什么访问权限？

解答：（1）用户 wenchang 启动进程 P 后，直到显示 China 时，P 的 EUID 对应 wenchang，EGID 对应 sisefellow，P 的 EUID 不等于 filex 的属主 weinan，P 的 EGID 不等于 filex 的属组 siselab_ms，而其余类用户在 filex 上没有任何权限，所以，进程 P 在显示 China 时，对文件 filex 没有任何访问权限。

（2）进程 P 在显示 England 时，程序映像已更新为 progf2，文件 progf2 打开了 SETGID 控制位，使 P 的 EGID 对应到 progf2 的属组 siselab_ms，该属组对 filex 拥有读权限，所以，进程 P 对文件 filex 拥有读的访问权限。

（3）进程 P 在显示 Australia 时，程序映像已更新为 progf3，文件 progf3 打开了 SETUID 控制位，使 P 的 EUID 对应到 progf3 的属主 weinan，该用户对 filex 拥有读和写权限，所以，进程 P 对文件 filex 拥有读和写的访问权限。

传统 UNIX 操作系统的自主访问控制机制根据“属主/属组/其余”式的访问控制思想，实现了基于有效用户属性的进程访问控制支持。

4.7　基于ACL的访问控制

基于“属主/属组/其余”的访问控制思想为操作系统中的访问控制提供了一种实用的方法，但存在明显的不足，那就是只能区分三类用户，粒度太粗。利用这种方法，针对给定的一个文件，难以做到为四个以上的用户分配相互独立的访问权限。访问控制表（Access Control List，ACL）思想可以为细粒度的访问控制提供比较好的支持。

4.7.1　ACL的表示方法

利用 ACL 机制，针对任意给定的一个文件，可以为任意个数的用户分配相互独立的访问权限。权限相互独立是指，改变分配给任意一个用户的权限，不会对其他用户的权限产生任何影响。

例 4.36　请对“属主/属组/其余”式的访问控制方法进行扩展，给出一个基于 ACL 的访问控制方法。

解答：把属主、属组和其余三个用户类扩展为属主、指定用户、属组、指定组和其

余五个用户类。其中，“指定用户”类可以包含任意个数的相互独立的用户。同样，“指定组”类可以包含任意个数的相互独立的用户组。

给每个文件定义一张表，用于存放 ACL 信息，表中的每一行定义一组访问权限，其中，“指定用户”类的每个用户占一行，“指定组”类的每个用户组占一行，其他每类用户各占一行。表的格式如下：

user:uname:RWX
group:gname:RWX
other::RWX

每行为一个记录，每个记录被冒号（:）分成三个字段。第一个字段是记录类型标记，user 标识用户记录，group 标识组记录，other 标识其余用户记录。第二个字段是名称，uname 表示用户名，用户名为空表示是属主，gname 表示组名，组名为空表示是属组。第三个字段表示权限，*R* 取值为“-”或“r”，对应“读”权限；*W* 取值为“-”或“w”，对应“写”权限；*X* 取值为“-”或“x”，对应“执行”权限。

以下是 ACL 表配置信息的示例：

user::rwx
user:wenchang:rwx
user:xiaoming:r-x
group::r-x
group:sisefellow:--x
other::r-x

示例中，第 1 行定义属主的权限，第 2、3 行分别定义用户 wenchang 和 xiaoming 的权限，第 4 行定义属组的权限，第 5 行定义组 sisefellow 的权限，第 6 行定义其余用户的权限。

根据例 4.36 给出的方法，对于任意一个文件 F，通过定义它的 ACL 表，可以根据需要，给任意用户 U 和任意组 G 独立地分配对它的访问权限，从而实现细粒度的访问控制。

例 4.37　用 ACL 机制进行访问控制，如果要支持“属主/属组/其余”式的访问控制功能，请问，最小的 ACL 表是什么样的？

解答：对于任意一个文件，最小的 ACL 表由以下配置行组成：

user::RWX
group::RWX
other::RWX

例 4.37 给出的 ACL 表的功能与 9 位的权限位串等价，其中，user 行定义文件的属主的权限，group 行定义文件的属组的权限，而 other 行定义其余用户的权限。由此，需

要注意一个问题，即文件属主和属组的名称（标识）都没有出现在 ACL 表中，如果需要属主或属组的标识，则必须到 ACL 之外去寻找。

为一个文件定义访问权限时，有时也需要对所有用户做一个总的限定。通过定义一个最大权限值，可以达到这个目的。

例 4.38　请以例 4.36 给出的 ACL 表为基础，给出一个能够在访问判定时限定所有用户的权限范围的简单方法。

解答：在 ACL 表中设立一个专用行，用于表示权限的最大值。即，只有其中定义了的权限，在访问判定时才有效；其中没有定义的权限，在访问判定时，将被过滤掉。用 mask 标识这个专用行，其格式是：

mask::RWX

访问判定时，先将用户的权限和 mask 行指定的权限进行“逻辑与”运算，再根据运算后得到的结果进行判定。

例 4.39　设文件 F 的 ACL 表中有如下设置：

user:wenchang:rwx

mask::r-x

请问，用户 wenchang 可以对文件 F 进行什么操作？

解答：因为“rwx” ⊕ “r-x” = “r-x”　（⊕ 表示“逻辑与”），所以用户 wenchang 可以对文件 F 进行读和执行操作。

在这个例子中，虽然给用户 wenchang 分配了 rwx 权限，但 mask 设置的权限值中没有 w（写）权限，所以，用户 wenchang 不能对文件 F 进行 w（写）操作。

在文件的 ACL 表中，并非一定要设置 mask 权限，一般来说，该设置是可选的，如果没有设置 mask 权限，则访问判定时，就不必进行权限过滤了。

4.7.2　基于ACL的访问判定

一个文件的 ACL 表中配置了很多用户和用户组对该文件的访问权限，其中包括文件的属主和属组的访问权限。判断一个进程能否对一个文件进行访问，基本思想应该是检查文件的 ACL 表中是否存在与进程相关的用户标识或组标识相匹配的表项，进而从中检查是否有符合条件的权限可用。

由于 ACL 表中配置了很多用户组，而一个进程的有效用户可能参加多个用户组，使一个进程可能与多个组关联，因此，ACL 表中可能有多个组与进程关联的多个组匹配，这给访问判定增加了一定的复杂性。

例 4.40　假定 ACL 表中没有 mask 表项，请给出一个根据 ACL 表判定进程访问文件的权限的方法。

解答：设 P 是一个任意的进程，F 是一个任意的文件，A 是一个任意的访问权限，需要判定的是 P 对 F 是否拥有 A 权限。按照以下过程进行判定。

（1）如果进程 P 的 EUID 等于文件 F 的属主的标识，那么，根据 F 的属主的 ACL 表项进行判定，转到第（7）步。

（2）如果进程 P 的 EUID 等于文件 F 的 ACL 表中某个指定用户的标识，那么，根据该指定用户的 ACL 表项进行判定，转到第（7）步。

（3）如果进程 P 的 XGID 等于文件 F 的属组的标识，而且，该属组拥有 A 权限，那么，根据 F 的属组的 ACL 表项进行判定，转到第（7）步。（XGID 是 P 的 EGID，或是 P 的 EUID 所属的某个用户组的标识。）

（4）如果进程 P 的 XGID 等于文件 F 的 ACL 表中某个指定组的标识，而且，该指定组拥有 A 权限，那么，根据该指定组的 ACL 表项进行判定，转到第（7）步。（XGID 的含义同（3）中的一致。）

（5）如果进程 P 的 XGID 等于文件 F 的属组的标识，或者等于 ACL 表中某个指定组的标识，但是，该属组和该指定组都不拥有 A 权限，那么，根据 F 的属组的 ACL 表项进行判定，转到第（7）步。（XGID 的含义同（3）中的一致。）

（6）根据文件 F 的 ACL 表中的 other 表项进行判定。

（7）如果选定的 ACL 表项中配置了 *A* 属性，那么，进程 *P* 对文件 *F* 拥有 *A* 权限，否则进程 P 对文件 F 不拥有 A 属性。

注意：例 4.40 中的第（3）或第（4）步完成时，已表明进程 P 对文件 F 拥有 A 权限，而第（5）步完成时，已表明进程 P 对文件 F 不拥有 A 权限。

例 4.41　假定 ACL 表中有 mask 表项，请给出一个根据 ACL 表判定进程访问文件的权限的方法。

解答：设 P 是一个任意的进程，F 是一个任意的文件，A 是一个任意的访问权限，需要判定的是 P 对 F 是否拥有 A 权限。假设无须对属主和其余用户进行权限过滤。在按照例 4.40 中的方法得出结果的基础上，再进行以下判定。

如果以下条件同时成立，那么，最后的结论是进程 P 对文件 F 不拥有 A 属性，否则保持例 4.40 得出的结论。条件是：

（1）例 4.40 得出的结论是进程 P 对文件 F 拥有 A 属性；

（2）例 4.40 在判定过程中用到的既不是属主表项也不是 other 表项；

（3）mask 表项中没有 A 属性。

IEEE 的 POSIX.1e 标准草案提供了 ACL 机制的一个规范，该规范规定对属主和其余用户不做 mask 限定，所以，在例 4.41 中采用了这一思想。

很多操作系统（尤其是 UNIX 类操作系统）都能提供 ACL 机制的支持。

4.8　特权分割与访问控制

在很多场合，管理和维护操作系统的用户都需要拥有一定的特权，才能顺利完成正常的系统服务工作。例如，如果操作系统的某个用户忘记了自己的口令，那该怎么办呢？采用常规途径，该用户将无法再通过原来的账户进入系统，只有采取特殊的措施，如删掉用户的口令，才能帮助用户恢复正常工作。采取特殊措施是需要特权的支持的。

4.8.1　特权的意义与问题

拥有特权的用户属于特权用户，而其他用户就是普通用户。UNIX 操作系统中的 root 用户就是典型的特权用户，实际上，它的名称是超级用户，具有最高的权限，可以完全不受操作系统的访问控制机制的约束。

前面说过，用户的工作实际上都是由进程代劳的，相应地，进程便有特权进程与普通进程之分。特权用户的工作由特权进程完成，普通用户的工作由普通用户完成。

特权是把双刃剑，既是系统服务之所需，也是系统威胁之所在。特权功能是操作系统安全性的隐患。以 UNIX 操作系统为例，如果攻击者获得了 root 用户的特权，他就获得了对整个系统的完全控制权，其后果不堪设想。

特权管理是操作系统安全性的重要内容。通过 4.1.1 节的介绍我们知道，J.H. Saltzer 和 M.D. Schroeder 于 1975 年提出的安全设计八大原则中的两大原则是面向特权的，它们是特权分离原则和最小特权原则。

特权分离原则就是要尽可能地对系统中的特权任务进行细分，让多个不同的用户去承担不同的细分任务，不要把系统特权集中到个别用户身上。就像财务工作中的情况那样，出纳管钱、会计管账、不要让同一个人既管钱又管账。

最小特权原则就是尽可能搞清楚完成某项特权任务所需要的最小特权，尽可能只给用户分配最小的特权，让他足以完成所承担的任务既可，也就是说，如果某项任务只需 n 项特权就能完成的话，不要给用户分配 n 项以上的特权。

4.8.2　特权的定义

特权分离原则与最小特权原则是密切相关的。只有在对系统中的特权进行合理划分的基础上，才有可能有效地实现最小特权原则。操作系统中的特权的定义是实现操作系统对最小特权原则的支持的难点之一。本节通过若干个例子来体会操作系统中的特权的定义方法。

例 4.42　请给出一个文件的查看方面的特权。

解答：定义一个记为 CAP_DAC_READ_SEARCH 的特权，使拥有该特权的用户在

读任何文件或目录时，都不会受到自主访问控制的“读”权限的限制。也就是说，就算用户没有对文件的读权限，也能对文件进行读操作。

例 4.43　请给出一个整个自主访问控制方面的特权。

解答：定义一个记为 CAP_DAC_OVERRIDE 的特权，使拥有该特权的用户在访问系统资源时，完全不受自主访问控制权限的限制。也就是说，就算用户对文件不拥有读、写、执行中的任何一个权限，也能对文件进行读、写和执行操作。

例 4.44　请给出一个文件属主权力方面的特权。

解答：定义一个记为 CAP_FOWNER 的特权，使拥有该特权的用户在对文件进行操作时，不受“必须是文件的属主”的限制。也就是说，就算用户不是文件的属主，只要其他条件符合，也能对文件进行只有文件属主才能进行的操作。

例 4.45　请给出一个操作系统重新启动方面的特权。

解答：定义一个记为 CAP_SYS_BOOT 的特权，使只有拥有该特权的用户才能进行操作系统的重启操作，没有该特权的用户不能进行相应的操作。

例 4.46　设操作系统支持可动态装载的内核模块，请给出一个装载内核模块方面的特权。

解答：定义一个记为 CAP_SYS_MODULE 的特权，使只有拥有该特权的用户才能把动态内核模块装入到操作系统中，或者删除操作系统中的动态内核模块，没有该特权的用户不能进行相应的操作。

例 4.47　请给出一个系统管理方面的特权。

解答：定义一个记为 CAP_SYS_ADMIN 的特权，使只有拥有该特权的用户才能进行安装文件系统、卸载文件系统、设置磁盘配额、设置主机名和域名、配置设备端口等方面的操作，没有该特权的用户不能进行相应的操作。

例 4.48　请给出一个网络管理方面的特权。

解答：定义一个记为 CAP_NET_ADMIN 的特权，使只有拥有该特权的用户才能进行网络接口配置、路由表修改、防火墙管理、代理服务的地址绑定等操作，没有该特权的用户不能进行相应的操作。

以上这些例子从若干个侧面介绍了操作系统中特权分割的思想。如果不进行类似的特权划分，就像在传统 UNIX 操作系统中那样，只要想进行其中的某项特权操作，如例 4.42 所需的操作，也只好给用户授予 root 超级用户特权了，可这样一来，用户就拥有了进行以上各个例中的所有操作的特权，显然不符合最小特权原则。

4.8.3　基于特权的访问控制

不论是基于权限位串的访问控制，还是基于 ACL 表的访问控制，都是基于用户标识的访问控制。基于用户标识的访问控制根据用户标识检索访问权限，进而确定访问判定结果。基于特权的访问控制与此不同，它根据特权来确定访问判定结果。

基于用户标识的访问控制定义了有效用户标识的概念，并把它作为访问控制判定的基础，它的值在进程的程序映像更新时进行确定。这些思想可以借鉴到基于特权的访问控制中来。

例 4.49　请给出一个支持基于特权的访问控制的基本体系结构。

解答：对任意一个可执行程序文件 F，为它设立一套特权集属性，用于保存特权集信息。用户根据程序 F 完成任务所需要的特权，为它进行特权配置，配置信息保存在 F 的特权集属性中。

对任意一个进程 P，为它设立一套特权集属性，用于存放特权集信息。进程 P 执行程序 F 时，根据 F 的特权集属性，建立进程 P 的特权集配置，存放到进程 P 的特权集属性中。

进程 P 进行访问操作时，根据进程 P 的特权集属性进行访问判定。

例 4.50　请给出一个在操作系统中确定文件和进程的特权集属性的方法，以便实现基于特权的访问控制。

解答：程序映像更新时的进程特权确定方法如图 4.10 所示。

为每个文件和进程设立三个特权集，分别是有效的特权集、许可的特权集和可继承的特权集，依次用 *E*、*P* 和 *I* 进行标记。*P* 特权集是可以分配给进程的所有特权的集合，*I* 特权集是进程演变时可以从旧进程继承到新进程的特权的集合，*E* 特权集是用于进行访问判定的特权的集合。

图 4.10 表示进程 $Proc_1$ 的程序映像通过 exec()系统调用更新为程序 F 后演变成进程 $Proc_2$ 的情形。E_0、P_0 和 I_0 是文件 F 的 *E* 特权集、*P* 特权集和 *I* 特权集。E_1、P_1 和 I_1 是旧进程 $Proc_1$ 的 *E* 特权集、*P* 特权集和 *I* 特权集。E_2、P_2 和 I_2 是新进程 $Proc_2$ 的 *E* 特权集、*P* 特权集和 *I* 特权集。另外，在系统范围内设立一个特权集上界 *B*。新进程的特权集的确定方法如下：

$$P_2 = (P_0 \ \& \ B) \mid (I_0 \ \& \ I_1)$$

$$E_2 = P_2 \ \& \ E_0$$

$$I_2 = I_1$$

其中，“&”和“|”分别表示“逻辑与”和“逻辑或”运算符。

当进程 $Proc_2$ 进行访问操作时，根据有效特权集 E_2 进行访问控制判定。

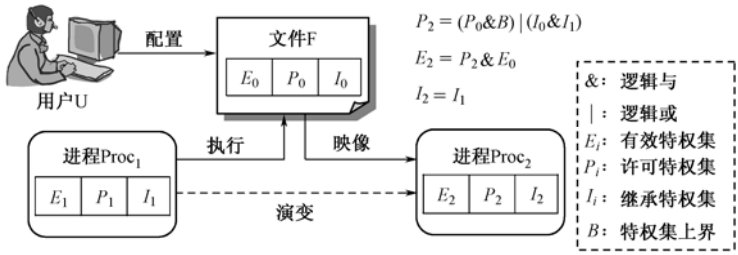


图 4.10　程序映像更新时的进程特权确定方法

例 4.50 给出了图 4.10 中进程 $Proc_2$ 的特权集的确定方法。进程 $Proc_1$ 的特权集如何确定呢？

如果进程 $Proc_1$ 是由别的进程通过 exec()系统调用演变得到的，那么，其特权集的确定方法与例 4.50 中给出的方法是类似的。如果进程 $Proc_1$ 是通过创建新进程的方式生成的，那么，其特权集与父进程的相同。至于系统中的第一个进程，其特权集则可由操作系统在初始化时确定。

IEEE 的 POSIX.1e 标准草案提供了一种称为权能（Capability）机制的规范，但该权能机制并不是传统意义上所说的与 ACL 机制相对的权能机制，而实际上是一种特权机制。该规范提供了对本节所介绍的特权思想的支持。

4.9　加密文件系统

访问控制机制是实现操作系统安全性的重要机制，但解决不了所有的安全问题。在操作系统正常工作的情况下，操作系统的访问控制机制能够有效地保护在操作系统控制范围之内的信息免遭非法访问，但是，一旦信息离开了某个操作系统的控制范围，该操作系统的访问控制机制对信息的安全性就无能为力了。

例如，不管计算机 A 上的操作系统的访问控制机制的功能多么强大，如果恶意用户能拿到计算机 A 的硬盘等存储信息的存储介质，他只需把存储介质安装到计算机 B 上，就能轻而易举地把信息读取出来，计算机 A 的访问控制机制此时鞭长莫及。

为了防止因信息载体落入他人之手而导致的信息泄露问题，可以采取对信息进行加密的措施。在操作系统中实现信息加密的方法很多，可以对单个文件进行加密，也可以对整个磁盘进行加密，本节以开放源码的 eCyptfs 为例，介绍介于两者之间的通过文件系统实现信息加密的方法。

定义 4.9　在操作系统中，具有对文件进行加密和解密功能的文件系统称为加密文件系统。

4.9.1　加密文件系统的应用方法

在操作系统中，信息以文件为单位进行处理，文件通过称为文件系统的数据结构进

行组织和管理。文件系统通常可以表示成由目录和文件构成的倒立的树型层次结构，如图 4.11 所示，这样的树型结构可简称为目录树结构。

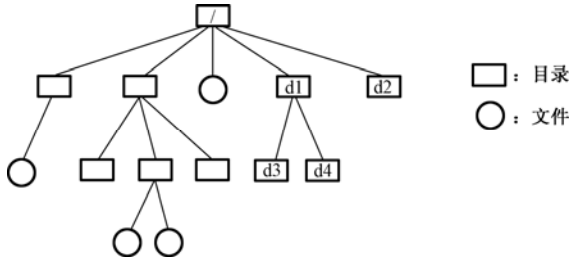


图 4.11　文件系统的树型层次结构

在目录树结构中，树的顶点是根目录（通常用斜杠“/”表示），根目录可以包含子目录和文件，子目录也可以包含子目录和文件，文件是叶节点。

面向单个文件的加密方法可以实现对任意给定的一个文件的加密，面向文件系统的加密方法则可以实现对任意给定的一个文件系统中的所有文件的加密。

在 UNIX 类操作系统中，一个文件系统 A 可以安装到另一个文件系统 B 中的一个目录上（该目录就称为安装点），使文件系统 A 的目录树成为文件系统 B 的目录树中的一棵子目录树。

按照这种思想，一个文件系统中的一个子目录结构对应的可能就是一个文件系统。因此，设计加密文件系统时，可以考虑面向子目录结构的文件加密，即对给定的任意一个子目录树中的所有文件进行加密。

例 4.51　请给出一个加密文件系统的文件加密应用框架，要求能够把对文件的加密处理限定在文件系统中任意给定的一棵子目录树中。

解答：在加密文件系统中实现一个约定：允许把文件系统中的任何一个目录定义为加密文件系统安装点，加密机制自动地对通过加密安装点写入到文件系统中的文件进行加密，自动地对通过加密安装点从文件系统中读出的文件进行加密。

例如，如果把图 4.11 中的目录/d2 定义为加密文件系统的安装点，把以目录/d1 为顶点的子目录树安装到该加密安装点上，并通过以下路径名创建以下文件和写入信息：

/d2/f0　①

/d2/d4/f1　②

/d2/d4/f2　③

/d2/d4/f3　④

/d2/d4/f4　⑤

则加密文件系统的加密机制自动实现对文件 f0、f1、f2、f3 和 f4 的加密，因为它们是通过加密安装点/d2 写入到文件系统上的。顶点为/d1 的子目录安装到加密安装点/d2 上如图 4.12 所示。

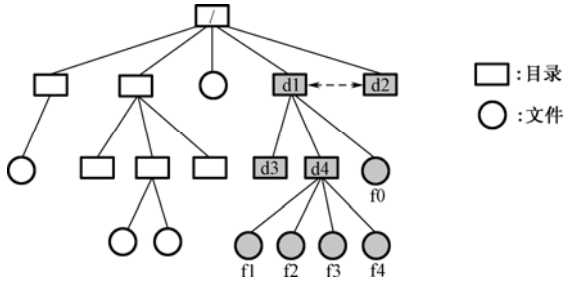


图 4.12　顶点为/d1 的子目录安装到加密安装点/d2 上

相应地，如果通过①～⑤中的路径名读文件 f0～f4，则加密文件系统的解密机制自动地对它们进行解密。

可以通过如下形式的一条命令把目录/d2 定义为加密文件系统的安装点，同时把以目录/d1 为顶点的子目录安装到该安装点上：

```
mount -t ecryptfs /d1 /d2
```

通过如下形式的一条命令则可以把已经安装到加密文件系统的安装点的以目录/d1 为顶点的子目录卸载掉：

```
umount /d1
```

子目录卸载后，加密文件系统就不再对其中的文件进行加密/解密处理了。

例 4.52　在图 4.11 所示的文件系统中，通过以下命令把目录/d2 定义为加密文件系统的安装点，同时把以目录/d1 为顶点的子目录安装到该安装点上：

```
mount -t ecryptfs /d1 /d2
```

通过以下路径名创建文件 f1：

/d2/d4/f1　　①

通过以下两个路径名都可以读文件 f1：

/d2/d4/f1　　②

/d1/d4/f1　　③

请问，通过以上两个路径名读到的文件结果是否相同，为什么？

解答：根据文件的创建方法知，f1 是一个经过加密的文件。通过以上两个路径名读到的文件结果是不相同的。因为，路径名②通过加密文件系统的安装点/d2 读文件 f1，所以，加密文件系统的解密机制自动对文件 f1 进行解密，因此读到的结果是明文。路径名③没有通过加密文件系统的安装点/d2 读文件 f1，所以，加密文件系统的解密机制不会自动对文件 f1 进行解密，因此读到的结果是密文。

例 4.53　在图 4.11 所示的文件系统中，通过以下命令把目录/d2 定义为加密文件系统的安装点，同时把以目录/d1 为顶点的子目录安装到该安装点上：

```
mount -t ecryptfs /d1 /d2
```

通过以下路径名创建文件 f1 和 f2：

/d2/d4/f1

/d1/d4/f2

按照以上两个路径名向文件 f1 和 f2 中写入相同的内容。请问：保存在文件系统中的文件 f1 和 f2 是否相同，为什么？

解答：文件 f1 是通过加密文件系统的安装点/d2 创建并写入内容的，因此，加密文件系统的加密机制自动对它进行加密，存放在文件系统中的文件 f1 是经过加密的文件。文件 f2 的创建及其内容的写入没有通过加密文件系统的安装点/d2，因此，加密文件系统的加密机制不会对它进行加密，存放在文件系统中的文件 f2 是没有经过加密的文件。所以，存放在文件系统中的文件 f1 和 f2 是不同的。

例 4.54　请利用例 4.51 给出的文件加密应用框架，给出一个加密文件系统的应用方法，要求解决例 4.52 和例 4.53 中存在的文件处理的不一致性。

解答：选择待安装的子目录的顶点为加密文件系统的安装点，则加密安装点与子目录顶点重合，对子目录中文件的访问就不存在不同的路径，从而消除文件处理的不一致性。

例如，对于图 4.11 所示的文件系统，可选择目录/d1 为加密文件系统的安装点，并借助如下命令把以目录/d1 为顶点的子目录安装到该安装点上：

```
mount -t ecryptfs /d1 /d1
```

结果如图 4.13 所示。这样，对顶点为/d1 的子目录中的文件的访问都必须通过加密安装点/d1，都得到加密文件系统的自动加密和解密处理，从而保证文件处理的一致性。

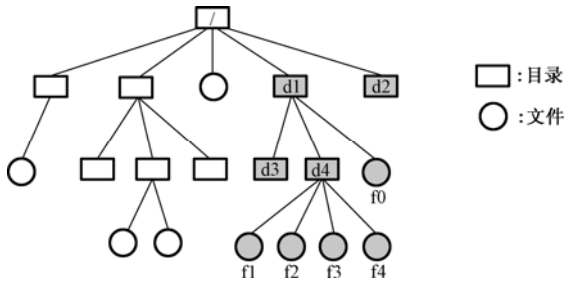


图 4.13　顶点为/d1 的子目录安装到加密安装点/d1 上

以上通过为加密文件系统定义安装点的方法，给出了加密文件系统的一个应用框架。即，经由安装点进行访问的文件，一律由加密文件系统进行加工处理；不经由安装点访问的文件，加密文件系统一概不进行加工处理。加工处理就是进行加密或解密，这样，用户只需在加密文件系统中定义一个安装点，并把一个子目录安装到该安装点上，系统就自动地对通过该安装点写入的文件进行加密，自动地对通过该安装点读出的文件进行解密。这就是 eCryptfs 加密文件系统支持文件的加密和解密应用的基本思想。

4.9.2　加密文件系统的基本原理

根据应用的需要，由用户任意指定文件系统中的一个子目录，让系统自动实现该子目录范围内所有文件的加密存储和解密使用，是一种灵活的文件系统加密方法，能够提供对应用透明的文件加密支持。

这样的文件系统加密功能可以通过在现有文件系统的基础上增加加密措施来实现，堆叠式文件系统（Stacked Filesystem）技术可以提供这方面的支持。堆叠式文件系统的核心思想是在现有文件系统之上叠加一层新的机制，从而为文件系统增加新的功能，比如，加密/解密功能。

例 4.55　请给出一个利用堆叠式文件系统技术实现加密文件系统的方案。

解答：eCryptfs 是一个在 Linux 操作系统中利用堆叠式文件系统技术实现的加密文件系统，该系统的基本结构和原理如图 4.14 所示。

eCryptfs 堆叠在 ext3 等现有文件系统之上，为 ext3 等现有文件系统增加文件的加密和解密功能。Linux 操作系统通过虚拟文件系统（VFS）为各类文件系统提供统一的应用接口，各类文件系统都处于 VFS 之下，堆叠式加密文件系统 eCryptfs 也不例外。从系统结构上看，eCryptfs 是插在 ext3 等现有文件系统与 VFS 之间的一个加密/解密处理层。

应用程序通过系统调用获得操作系统的文件访问服务。应用程序发出的访问文件的系统调用启动 VFS 的工作，VFS 进行相应处理后，启动 etx3 或其他具体的文件系统进行实际的文件操作。

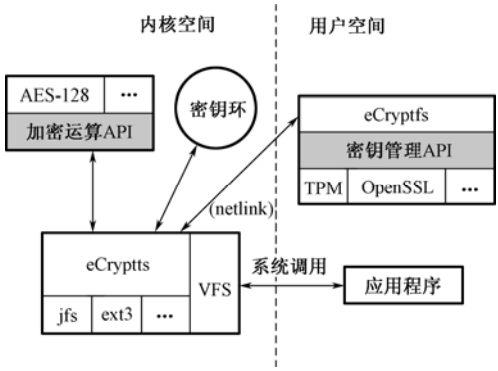


图 4.14　加密文件系统 eCryptfs 的基本结构和原理

插入了 eCryptfs 加密/解密处理层之后，如果用户要访问的是加密安装点之下的文件，则 VFS 首先启动 eCryptfs 进行文件的加密/解密处理，然后再由 eCryptfs 启动 ext3 或其他文件系统对文件进行处理。

应用程序读文件时，ext3 或其他文件系统从磁盘上读出文件，并交给 eCryptfs，eCryptfs 将文件解密，然后交给 VFS，最后，VFS 把文件传给应用程序。应用程序写文件时，VFS 把应用程序提供的文件交给 eCryptfs，eCryptfs 将文件加密后再交给 ext3 或其他具体的文件系统，具体的文件系统把文件写入到磁盘上。

可见，在 eCryptfs 加密文件系统的作用下，存放到磁盘上的文件是经过加密的，不过，这对应用程序来说是透明的，应用程序处理的是不加密的文件，eCryptfs 加密文件系统存在与否对应用程序的工作方式没有任何影响。

一个堆叠式文件系统并不是一个完整的文件系统，而只是一个堆叠在现有文件系统之上的一个功能层，eCryptfs 是一个典型的例子，它堆叠在 ext3 等特定文件系统之上，提供文件的加密/解密功能，把特定文件系统扩充为具有文件加密/解密功能的文件系统，实现文件的加密存储。

经过 eCryptfs 处理后，存储到磁盘上的文件是加过密的文件，就算磁盘落到攻击者手中，文件的内容也不容易泄露，这为文件的安全保存和传输提供了有效的支持。合法用户只要在 eCryptfs 的工作环境中对文件进行正常操作，加过密的文件会被系统自动解密还原成明文形式，因此，用户可以正常地使用它们。

4.9.3　加密算法的加密密钥

堆叠式加密文件系统的结构为指定的子目录中所有文件的透明加密奠定了很好的基础，落实到每个文件的加密操作上，加密算法和加密密钥的使用是关键。

eCryptfs 加密文件系统堆叠在 ext3 等低层特定文件系统之上实现文件的加密。低层特定文件系统很多，为叙述简单起见，本节以 ext3 文件系统为例进行说明。

1．文件的加密与解密

图 4.15 描述了 eCryptfs 加密文件系统对一个文件进行加密/解密处理的方法。一个文件在 eCryptfs 加密文件系统中有两个视图，在 eCryptfs 层，是 eCryptfs 格式的文件视图，这是明文的文件视图，在 ext3 层，是 ext3 格式的文件视图，这是密文的文件视图。

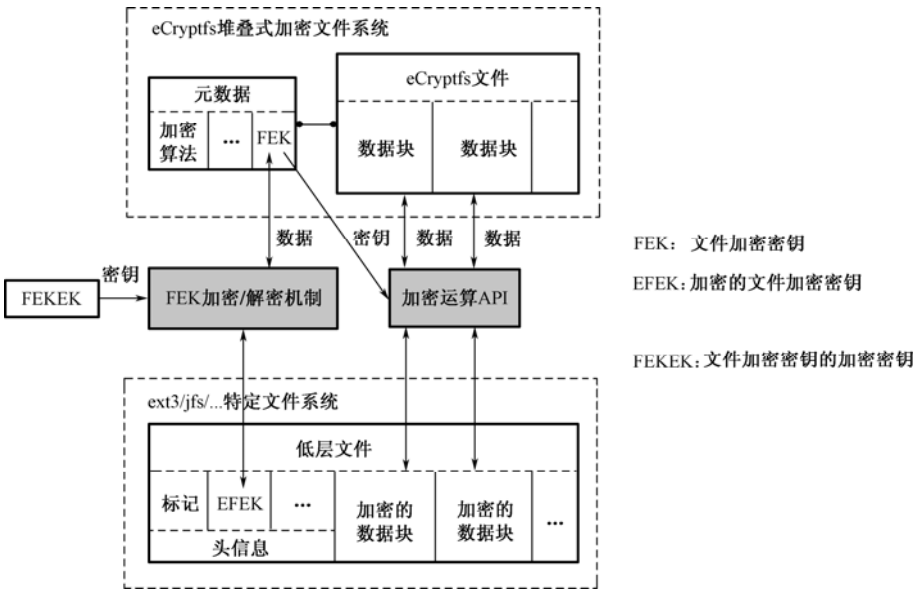


图 4.15　加密文件系统 eCryptfs 对文件的加密/解密方法

应用程序操作的是明文的文件，明文的文件经 VFS 传递到 eCryptfs 中时，仍然以明

文的形式存在。eCryptfs 把明文的文件加密后再传递给 ext3 文件系统，所以，文件在 ext3 文件系统中以密文的形式存在，这就是 eCryptfs 实现文件加密所要达到的效果。

eCryptfs 加密的是文件的内容，文件的属性并不加密，描述文件属性的 i 节点中的各种信息仍以明文的形式存在。在 eCryptfs 文件系统层，eCryptfs 把文件的内容划分为数据块，对数据块进行加密，然后把加密的数据块传递到 ext3 文件系统层，由 ext3 文件系统把它们存储到磁盘上。

与在 eCryptfs 层的文件内容相比，在 ext3 文件系统层，文件的内容中增加了头信息。顾名思义，头信息位于文件内容的头部，其作用是描述文件内容的加密方法。重要的头信息包含用于对文件进行加密的密钥及用于对该密钥进行加密的密钥的相关信息。

Linux 操作系统内核提供 AES 等加密功能函数和加密运算 API，eCryptfs 通过加密运算 API 调用 Linux 内核中的加密运算功能实现对文件的加密，如图 4.14 所示。

处理新文件时，eCryptfs 为文件生成加密密钥（FEK，File Encryption Key），利用 FEK，通过加密运算 API 调用内核的加密函数，即可实现对文件内容数据块的加密。FEK 是调用内核的随机数生成函数生成的随机数。eCryptfs 把 FEK 等信息组织成元数据，用于在 eCryptfs 层描述文件的加密方法。

如图 4.15 所示，eCryptfs 层的元数据与 ext3 层的头信息相对应，文件的加密密钥作为头信息的一部分嵌入在文件的内容之中，随文件一起保存和传输。当然，嵌入在文件内容之中的文件加密密钥不是明文，是经过加密的文件加密密钥（EFEK，Encrypted FEK）。

用于对 FEK 进行加密的密钥就是文件加密密钥的加密密钥（FEKEK，FEK Encryption Key）。即，以 FEKEK 为密钥，调用内核的加密函数，对 FEK 进行加密，得到 EFEK。EFEK 保存在加密文件的头信息中。

处理新文件时，根据 eCryptfs 层的元数据生成 ext3 层的头信息，其间，用 FEKEK 把 FEK 加密成 EFEK；处理旧文件时，根据 ext3 层的头信息生成 eCryptfs 层的元数据，其间，用 FEKEK 把 EFEK 解密成 FEK。

写文件时，eCryptfs 把应用程序提供的文件的内容划分成为数据块，组织成 eCryptfs 格式的文件内容，以 FEK 为密钥，对数据块进行加密，加密后的数据块按照原来的顺序组合在一起，加上头信息，组织成 ext3 格式的文件内容，由 ext3 存放到磁盘上。

读文件时，ext3 从磁盘上读出 ext3 格式的文件内容，eCryptfs 从 ext3 格式的文件内容中分离出头信息和加密的数据块，根据头信息可以得到 FEK，以 FEK 为密钥，对加密的数据块进行解密，解密后的数据块按照原来的顺序组合在一起，形成 eCryptfs 格式的文件内容，eCryptfs 把该文件内容作为一个整体提供给应用程序，应用程序便得到它所需要的文件内容。

2．加密密钥的加密密钥

那么，如何确定 FEKEK 呢？下面通过一个例子来讨论这个问题。

例 4.56　在 eCryptfs 加密文件系统中，文件的加密密钥 FEK 是随机生成的，eCryptfs 以 FEKEK 为密钥把它加密成 EFEK 后嵌入到加密文件内容的头信息中，请给出一个确定 FEKEK 的方法。

解答：让用户在把子目录安装到加密文件系统的安装点上时提供一个口令 D_{pw}，根据口令 D_{pw} 生成 FEKEK。设 A_{MD5} 是 MD5 哈希算法，A_{random} 是一个随机数生成算法，按照以下步骤确定 FEKEK：

（1）生成一个盐值：$D_{salt} = A_{random}()$；

（2）把盐值撒到口令上：$D_{tmp} = D_{pw} \| D_{salt}$；

（3）循环 65535 次执行这个运算：$D_{tmp} = A_{MD5}(D_{tmp})$；

（4）确定密钥：$FEKEK = D_{tmp}$。

当用户把子目录安装到加密安装点时，按照以上方法生成 FEKEK，并把它存放到内核的密钥环中，如图 4.14 所示。当 eCryptfs 需要对文件进行加密/解密操作时，从密钥环中即可找到 FEKEK。

例 4.56 给出的是一个基于口令的文件加密密钥的加密密钥 FEKEK 的生成方法，采用这种方法，用户必须提供正确的安装口令，才能对加密文件进行解密操作。也可以采用其他方法生成 FEKEK，通过用户空间中的密钥管理 API，eCryptfs 为多种密钥生成方法提供支持。

例 4.57　在 eCryptfs 加密文件系统中，FEKEK 是关键的密钥，有了它就可以解密 EFEK 得到 FEK，从而解密得到明文的文件，如果采用基于口令的 FEKEK 生成方法，请给出一个存储和验证 FEKEK 相关信息的方法。

解答：设 A_{MD5} 是 MD5 哈希算法，用以下运算确定 FEKEK 的密钥标识 I_{FEKEK}：

$$I_{FEKEK} = A_{MD5}(FEKEK)$$

把密钥标识 I_{FEKEK} 存放在加密文件内容的头信息中。

用户安装子目录时，根据用户提供的口令生成密钥标识 I_{FEKEK}。eCryptfs 打开加密文件时，从文件内容的头信息中取出密钥标识 I_{FEKEK}。对比新生成的 I_{FEKEK} 标识与保存在文件中的 I_{FEKEK} 标识，如果两者相等，则验证成功，否则验证失败。

根据例 4.57 给出的方法，只需保存 FEKEK 的标识信息，无须保存口令信息，只要通过计算，就能验证口令的正确性。实际上，该例的方法只用到 FEKEK 的值，与 FEKEK 的生成方法无关，对于其他非口令式的 FEKEK 生成方法，同样可以按照该例的方法存储和验证密钥标识信息。

4.10　系统行为审计

前面各节介绍的身份认证、访问控制、特权管理和文件加密等机制是操作系统中重

要的安全机制。但是，仅仅建立安全机制还说明不了问题，它们必须正常工作，才有可能发挥安全支持作用。而且，绝对的安全是不存在的，安全问题在所难免。

那么，如何知道系统的安全机制是否正常工作呢？如何判断系统是否已经出现了安全问题呢？如何确定已发生的安全问题对系统造成的危害呢？这需要对系统中的安全相关行为进行监测，这种系统安全行为的检测过程通常称为审计（Auditing）。

审计的关键是建立和分析系统的行为记录信息，这样的行为记录称为审计记录（Audit Trail），也称为日志（Log）。在很多场合，审计与日志处理（Logging）作为同义的概念使用，因为日志处理是系统审计的重要手段。

正像沙子记录着从海滩走过的动物的脚印一样，日志信息记录了系统行为的历史足迹。审计有助于了解系统安全机制的工作状况，有助于分析系统中出现的问题或受到的攻击，有助于实现系统出现异常后的恢复，它在计算机取证（Forensics）中也具有重要的意义。

UNIX 操作系统的 syslog 审计服务提供丰富的日志信息处理功能，本节通过它来了解系统审计的基本方法。

4.10.1　审计机制的结构

UNIX 操作系统的 syslog 审计服务系统提供通用的日志处理服务，系统的基本结构如图 4.16 所示，系统由 syslogd 日志处理进程、配置文件、日志文件和 syslog 函数库等主要部分构成。

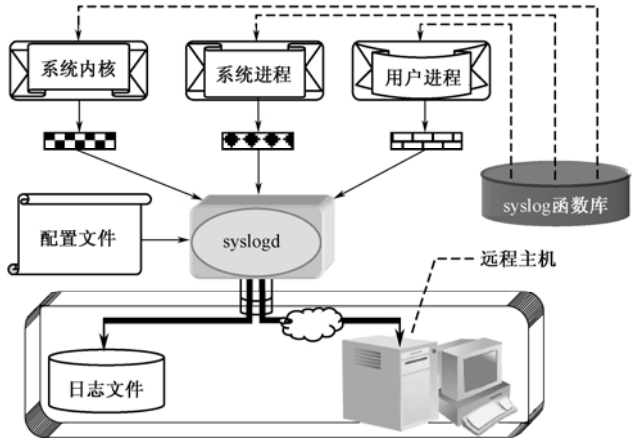


图 4.16　syslog 审计服务系统的结构

syslogd 是一个守护进程，它接收系统中各种进程发送的审计事件，根据配置文件中设定的指令，统一生成日志信息，并送到指定的目的地中。目的地形式多样，由配置文件指定。通常，目的地是本地的日志文件，在这种情况下，syslogd 守护进程把日志信息保存到本地的日志文件中。目的地也可以是网络中的其他主机，通过这种方式，可以把

日志信息保存到独立的专用日志主机上，提高日志信息的安全性，就算本地主机被攻陷，日志信息也能免遭破坏，确保出现安全问题后的事后分析能够顺利进行。目的地还可以是系统控制台、指定的用户或指定的进程。

操作系统中的系统进程、用户进程及系统内核都有可能产生审计事件，它们可以调用 syslog 函数库中的函数把审计事件及相关的信息发送给 syslogd 守护进程，由 syslogd 为它们生成并处理日志信息。

4.10.2　审计指令的配置

syslog 审计服务系统的配置文件是/etc/syslog.conf，这是一个文本文件，其中定义了指导 syslogd 守护进程工作的指令。

例 4.58　以下是/etc/syslog.conf 配置文件中的部分配置信息（行首的行号是为了便于说明而添加的，实际文件中没有），请说明它们的主要含义。

```
1 *.err;kern.debug;auth.notice    /dev/console
2 daemon,auth.notice              /var/log/messages
3 auth.*                          root,wenchang
```

解答：每行配置信息由两个域构成，第一个域（左侧部分）描述日志信息源，第二个域（右侧部分）描述日志信息目的地。

信息源一方面说明日志信息是由什么产生的，另一方面说明日志信息属于哪个紧迫级别。信息目的地说明日志信息将发往何处。

第 1 行中的“auth.notice”表示由认证进程产生的 notice 及其以上级别的日志信息，“kern.debug”表示由操作系统内核产生的 debug 及其以上级别的日志信息，“*.err”表示所有 err 及其以上级别的日志信息。该行的信息源域说明了由分号分隔的三类信息。该行的目的地域说明的是系统控制台。该行说明的三类日志信息将发送到系统控制台。

第 2 行的信息源域说明的是由系统守护进程和认证进程产生的 notice 及其以上级别的日志信息，其中，daemon 表示系统守护进程，auth 表示认证进程，两者间用逗号分隔。该行的目的地域说明的是/var/log/messages 日志文件。该行说明的两类日志信息将保存到指定的日志文件中。

第 3 行的信息源域说明的是由认证进程产生的所有级别的日志信息，该行的目的地域说明的是 root 用户和 wenchang 用户。该行说明的日志信息将发送给指定的两个用户。当 syslogd 进程给指定用户发送指定类型的日志信息时，如果用户已登录到系统中，则可以看到相应的信息，否则看不到信息。

由例 4.58 可知，syslog 审计服务系统给日志信息定义了紧迫级别，实际上，就是相应审计事件的紧迫级别。syslog 为日志信息共定义了 8 个紧迫级别，由高到低，如表 4-2 所示。

表 4-2　syslod 日志信息的紧迫级别

紧 迫 级 别	解　　释
emerg	紧急：出现了紧急情况。例如，系统即将崩溃，此类信息通常向所有用户广播
alert	警告：发生了需要立刻改正的事情。例如，系统数据库遭到破坏
crit	关键：出现了关键问题。例如，硬件出错
err	出错：普通错误信息，表示遇到了不正确的事情
warning	提醒：遇到非常规情况
notice	注意：遇到特殊事情，不是错误，但可能需要特殊处理
info	通知：一般消息
debug	提示：调试程序时使用的信息

表 4-2 中的解释不是绝对的，要把握住的主要是各个紧迫级别之间的高低次序。对照该表去理解例 4.58，其中含义将更加明了。

根据例 4.58 的介绍可知，日志信息源的描述包含两种成分，其一是产生日志信息的主体，其二是信息的紧迫级别。表 4-3 给出了 syslod 中产生日志信息的主要主体。

表 4-3　syslod 中产生日志信息的主要主体

主 体 类 别	解　　释
auth	认证进程，即要求用户提供用户名和口令的进程，如 login 等
authpriv	涉及特权信息的认证进程
kern	操作系统内核
cron	执行周期性任务的 cron 守护进程
daemon	系统的其他守护进程
mail	电子邮件系统
ftp	FTP 文件传输系统
syslog	syslogd 审计服务守护进程
lpr	打印系统
local0... local7	留做系统定制用途，如 Fedora 系统用 local7 表示系统引导程序
mark	每隔一定时间（如 20 秒）发送一次消息的时间戳服务进程
news	网络新闻系统（usenet 等）
user	普通用户进程

例 4.59　以下是/etc/syslog.conf 配置文件中的部分配置信息（行首的行号是为了便于说明而添加的，实际文件中没有），请说明它们的主要含义。

```
1 auth.*          @sise.ruc.edu.cn
2 authpriv.*      @loghost
3 mail.*          -/var/log/maillog
4 cron.*          /var/log/cron
5 *.emerg         *
6 mark.*          /dev/console
```

```
7 local7.*          /var/log/boot.log
8 news.=crit        /var/log/news/news.crit
9 *.alert           |dectalker
```

解答：例 4.58 说明过的内容这里不再重复，着重说明不同的内容。首先看日志信息的目的地域。第 1 行把信息发给 sise.ruc.edu.cn 远程主机。第 2 行把信息发给用别名 loghost 表示的远程主机，这比第 1 行的表示灵活，改变主机名时，只需修改别名定义，无须修改配置文件。第 3 行把信息保存到指定日志文件中，日志文件路径名前的减号表示允许信息缓冲，即产生日志信息时不必立刻写入日志文件，这能提高系统性能，但存在日志缺失的可能。第 5 行把信息发给所有用户。第 9 行通过有名管道把信息发给进程 dectalker 进一步处理。

现在看日志信息的信息源域。第 1、2、3、4、6、7 行分别表示认证进程、涉及特权信息的认证进程、邮件系统、cron 进程、时间戳进程、系统引导程序产生的所有级别的日志。第 5 行表示 emerg 级别的所有信息。第 9 行表示 alert 及以上级别的所有信息。第 8 行表示新闻系统产生的 crit 级别的信息。

syslog 审计服务系统的配置文件通过日志信息源与日志目的地的对应关系为 syslogd 进程定义工作指令，指示它遇到某审计主体产生某紧迫级别的审计事件时把生成的日志送往指定的目的地。审计主体主要有操作系统引导程序、操作系统内核及系统中的进程等。审计事件的紧迫级别就是日志信息的紧迫级别。

4.10.3　审计信息的分析

syslog 审计服务系统的日志信息主要存放在/var/log 目录下的日志文件中，表 4-4 列出了该目录下的一些典型的日志文件。

表 4-4　/var/log 目录下的典型日志文件

日志文件名	解　　释
dmesg	内核启动日志：系统引导时，内核产生的信息
boot.log	系统引导日志：哪些系统服务已成功启动或关闭，哪些不能成功启动或关闭
secure	安全日志：用户登录尝试和会话的日期、时间和持续时间等方面的信息
messages	通用日志：很多进程产生的日志保存在其中
cron	cron 日志：crond 进程的状态信息，它周期性地执行时间表中的任务
httpd/access_log	Apache 访问日志：记录向 Apache Web 服务器获取信息的请求
maillog	电子邮件日志：记录邮件的发件人和收件人等方面的信息
sendmail	sendmail 日志：记录 sendmail 进程产生的出错信息
mysqld.log	MySQL 服务日志：记录 MySQL 数据库服务器的活动
vsftpd.log	FTP 服务日志：记录由 vsFTPd 服务进程执行的 FTP 文件传输活动

日志文件包含日志信息，日志信息由日志记录（审计记录）组成，每个日志记录对

应一行日志信息。构成日志记录的主要内容有：

- 产生日志记录的日期和时间；
- 产生日志信息的计算机的名称；
- 产生日志信息的程序或服务的名称；
- 产生日志信息的进程的进程号；
- 日志信息正文。

例 4.60　以下是 syslog 系统的/var/log/messages 日志文件的部分内容，请简要说明其中的含义。

Aug 21 11：04：32 siselab network：Bringing up loopback interface：succeeded

Aug 21 11：04：35 siselab network：Bringing up interface eth0：succeeded

Aug 21 13：01：14 siselab vsftpd(pam_unix)[10565]：authentication failure；logname= uid=0 euid=0 tty= ruser= rhost=61.135.170.110 user=alice

Aug 21 14：44：24 siselab su(pam_unix)[11439]：session opened for user root by alice(uid=600)

解答：该内容共包含四条日志记录，它们的生成日期都是 8 月 21 日。第一条记录的生成时间是 11 点 4 分 32 秒，最后一条记录的生成时间是 14 点 44 分 24 秒。第一条和第二条记录表明网络成功启动。第三条记录表明 IP 地址为 61.135.170.110 的计算机上的用户试图以用户名 alice 登录 FTP 服务器，认证失败。第四条记录表明用户 alice 执行 su 命令成功地将身份转换成用户 root。

经常分析日志文件中的信息，有助于及早发现攻击者对系统进行攻击的企图。例如，如果日志信息显示出系统中存在着对某个特定服务的大量反复连接尝试，特别是当这些连接尝试来自 Internet 时，系统就很有可能正遭受攻击。发现这种情况后，便可采取积极的应对措施，以确保系统的安全。

4.11　本章小结

本章在勾画操作系统安全性全景视图的基础上，主要从身份标识与认证的基本方法、面向网络的身份认证、基于 PAM 的认证框架、基于权限位的访问控制、进程的有效身份与权限、基于 ACL 的访问控制、特权分割与访问控制、加密文件系统和系统行为审计等方面介绍操作系统的基础安全性。

操作系统安全性研究的历史源远流长，其基本理论、技术和方法在 20 世纪 60～70 年代逐步建立起来，此后，持续不断地向前发展。操作系统安全性的内容非常丰富，内存保护、客体重用、身份标识与认证、访问控制、可信路径、加密支持、特权管理、安全审计、多安全策略的灵活支持、隐蔽信道处理及完整性与可用性保护等是其中的重要组成部分。操作系统安全性在信息系统安全中具有不可替代的作用。

在很多情形下，操作系统的安全控制与用户的身份有密切的关系，合法用户身份的

标识和认证是操作系统安全控制的基础。操作系统为合法用户设立账户，以实现对合法用户的标识和管理。最简单且最常用的身份认证方法是利用口令进行认证。给口令撒盐能提高破解口令的难度，撒盐就是给口令掺入随机数，掺入的随机数称为盐值。实现口令信息与账户信息的分离能够提高口令信息的安全性。

把身份认证信息组织起来，集中存放到服务器中，借助服务器实现身份认证信息的共享，可以解决用户在网络中不同主机上的身份认证问题。把公开密钥密码算法和秘密密钥密码算法结合起来，可以设计出网络中的身份认证和信息加密传输方法。仅采用秘密密钥密码算法，通过基于通行证和服务卡二次认证框架，可以实现在加密传输基础上的用户身份认证和服务请求认证。

PAM 是一个统一的身份认证框架，它由 API、动态共享库和配置文件构成，它通过一个插拔式的接口实现服务程序与认证机制的随意组合，实现服务程序与认证机制的分离，从而简化认证系统的管理和维护工作，增加认证系统的灵活性。

考虑用户对文件的操作，可以定义读、写、执行三种方式，考虑用户访问文件的许可，可以定义读、写、执行三种权限，站在文件的角度，可以把用户划分为属主、属组和其余三种类型，利用一个 9 位的二进制位串，可以表示三类用户对文件的三种访问权限，从而实现基于权限位的“属主/属组/其余”式的访问控制。

进程是用户的化身，用户对系统的操作由进程代为实施。进程必须拥有访问权限才能对文件进行访问，可以为进程定义用户属性，用做访问判定的依据。对进程的访问控制，使用进程的有效用户标识（EUID）和有效用户组标识（EGID）属性进行判定。进程的用户属性在进程创建和程序映像更新时被确定。

利用 ACL 机制，对于任意一个文件 F，通过定义它的 ACL 表，可以根据需要，给任意用户 U 和任意组 G 独立地分配对它的访问权限，从而实现细粒度的访问控制。判断一个进程 P 能否对一个文件 F 进行访问的方法是检查文件 F 的 ACL 表中是否存在与进程 P 的用户属性匹配的项，进而检查是否有符合条件的权限。

特权分离原则要求尽最大可能对系统中的特权进行细分，最小特权原则要求尽最大可能给用户分配最小的特权。基于用户标识的访问控制根据用户标识确定访问判定结果，基于特权的访问控制根据特权来确定访问判定结果。可以给文件和进程定义特权集属性，用于实现基于特权的访问控制。

加密文件系统可以采取面向子目录的文件加密处理方法，允许把任意目录定义为加密安装点，规定经由该安装点进行访问的文件，一律进行加密处理，不经由该安装点访问的文件，一概不做加密处理，只需把一个子目录安装到加密安装点上，系统就自动地对通过该安装点写入的文件进行加密，自动地对通过该安装点读出的文件进行解密，从而实现对应用透明的文件加密处理。

对系统中的安全相关行为进行监测的过程称为审计，审计的关键是建立和分析系统的行为记录信息。UNIX 操作系统的 syslog 审计服务系统由 syslogd 日志处理进程、配置

文件、日志文件和 syslog 函数库等主要部分构成。syslogd 进程接收系统中各种进程发送的审计事件，根据配置文件中设定的指令统一处理日志信息。经常分析日志文件中的信息有助于及早发现攻击者对系统进行攻击的企图，进而确保系统的安全。



习题 4

1. 操作系统安全性包括哪些主要内容？它们主要解决什么问题？

2. 什么是自主安全性？什么是强制安全性？自主安全性与自主访问控制之间、强制安全性与强制访问控制之间分别是什么关系？

3. 什么是可信路径？

4. 试论述为什么操作系统安全性对于解决信息系统中的安全问题具有不可替代的作用。

5. 操作系统中最基本的身份认证方法是账户名加口令的方法，操作系统通常把账户名和口令相关信息保存在账户信息数据库文件中。请问，在文件中保存口令相关信息的常见方法有哪些？试分析它们的安全性。

6. 在口令中撒盐是什么意思？试举例说明在 UNIX 系统实现口令撒盐的基本方法。

7. 试举出操作系统支持的三种常见的网络化用户身份认证方法，并说明它们的基本原理和应用方法。

8. 操作系统中的 PAM 身份认证框架是如何为具有身份认证需求的服务程序提供灵活的身份认证支持的？该框架由哪些主要部分构成？为了使该框架发挥作用，系统管理员需要完成哪些工作？

9. 已知 Linux 操作系统实现的是“属主/属组/其余”式的自主访问控制机制，系统中部分用户组的配置信息如下：

gteacher:x:850:utone,uttwo,utthree,utfour

gstudent:x:851:usone,ustwo,usthree,usfour

文件 fone 的部分权限配置信息如下：

rw---xr-x　utone gteacher　..... fone

请解答以下问题：

（1）用户 utone、uttwo 和 usone 对文件 fone 分别拥有何种访问权限？

（2）用户 utone 是否有办法执行文件 fone？

10. 已知 Linux 操作系统中的部分用户组的配置信息如第 9 题所示，部分文件的权限配置信息如下：

--x--x--x	utone	gteacher	 progf1
--s--x--x	usone	gstudent	 progf2
--x--s--x	ustwo	gstudent	 progf3
rwxr-x--x	usone	gstudent	 filex

设 progf1、progf2 和 progf3 这三个可执行程序的伪代码如下：

```
progf1：
        printf( “China” );
        exec( “progf2” );
progf2：
        printf( “England” );
        exec( “progf3” );
progf3：
        printf( “Australia” );
        return;
```

用户 utone 执行程序 progf1 启动了进程 P，请问进程 P 在显示 China、England、Australia 时，对文件 filex 分别拥有什么访问权限？

11. 与“属主/属组/其余”式的访问控制机制相比，ACL 访问控制机制有什么优点？

12. 设有一个实现了 POSIX.1e 的权能机制的 Linux 系统，该系统中的进程 P 在 *T* 时刻执行了程序文件 progf，试说明根据进程 P 在 *T* 时刻之前的特权属性和程序文件 progf 的特权属性确定进程 P 在 *T* 时刻之后所拥有的特权的方法。

13. 试简要说明 eCryptfs 加密文件系统的基本工作原理和设计原理。

14. 如图 4.12 所示，以目录/d2 为 eCryptfs 加密文件系统的安装点，用以下命令把子目录/d1 安装到该安装点上：

```
mount -t ecryptfs /d1 /d2
```

创建空文件 F1 和 F2，使它们的路径名分别为：

```
F1 = /d1/d4/f3
F2 = /d2/d4/f4
```

分别向 F1 和 F2 中写入字符串“北京欢迎您!”，并设：

```
F3 = /d1/d4/f4
```

现通过电子邮件把 F1 和 F3 发送给远端用户 U，请问用户 U 看到的与文件 F1 和 F3 对应的内容分别是什么？

15. 简要说明 UNIX 操作系统中的 syslog 审计服务系统的基本结构和工作原理。

第 5 章
操作系统的增强安全性

第 4 章介绍了操作系统的基础安全性方面的知识，其中，在访问控制方面，主要介绍了自主访问控制的基本方法。本章从强制访问控制方面讨论操作系统的增强安全性。强制访问控制以强制访问控制模型为支撑。本章通过支持强制访问控制的域类实施（DTE，Domain and Type Enforcement）模型，以 SELinux 操作系统的强制访问控制机制的实现为例，探讨操作系统强制访问控制的基本技术和方法，主要内容包括 TE 模型与 DTE 模型、SELinux 实现的 TE 模型、访问判定与切换判定、SELinux 的系统结构设计和 SELinux 的策略语言等。

5.1　TE模型与DTE模型

按类实施（TE，Type Enforcement）模型是一个强制访问控制模型，该模型最初是由 W.E. Boebert 和 R.Y. Kain 于 1985 年提出的，是为安全 Ada 目标（Secure Ada Target）系统设计的一个安全模型。安全 Ada 目标系统后来更名为逻辑协处理内核（LOCK，Logical Coprocessing Kernel）系统。1991 年，R. O'Brien 和 C. Rogers 为 LOCK 系统丰富了 TE 模型的内容。1994 年，L. Badger 和 D.F Sterne 对 TE 模型进行了改进，得到了 DTE 模型。1995 年，L. Badger 等人在 UNIX 操作系统中实现了 DTE 模型。下面首先介绍 TE 模型的基本思想，继而介绍 DTE 模型的基本思想。

5.1.1　TE模型的基本思想

TE 模型实现强制访问控制的基础是对主体和客体进行分组，并定义了域和类型的概念。

定义 5.1　在 TE 模型中，把系统中的所有主体划分成若干组，每一个组称为一个域（Domain），把系统中的所有客体划分为若干组，每一个组称为一个类型（Type）。

根据以上定义，每一个主体都有一个域与它对应，为了描述主体与域之间的对应关系，给每个主体定义一个域标签（Label）。同样，为了描述客体与类型之间的关系，给每个客体定义一个类型标签。主体的域标签和客体的类型标签是 TE 模型中的访问控制属性，TE 模型的访问授权方法要确定的是域对类型所拥有的访问权限。

TE 模型是面向二维表的访问控制模型，具有如下定义。

定义 5.2　在 TE 模型中，用于描述域与类型之间的访问授权关系的二维表称为域定义表（DDT，Domain Definition Table），表中的行与域对应，表中的列与类型对应，行与列的交叉点所对应的元素表示相应的域拥有的对相应的类型的访问权限。

TE 模型的域定义表如图 5.1 所示。

在图 5.1 中，域 D_i 拥有的对类型 T_j 的访问权限由元素 A_{ij} 表示，元素 A_{ij} 给出的是

读、写、执行等访问权限的集合。

一个域通常包含多个主体，一个类型通常包含多个客体，在图 5.1 中，域 D_i 包含 P_1、P_2、…、P_x 等进程，类型 T_j 包含 F_1、F_2、…、F_y 等文件。

判定主体对客体的访问权限时，由主体的域标签确定相应的域，由客体的类型标签确定相应的类型，根据相应的域定位 DDT 表中的行，根据相应的类型定位 DDT 表中的列，根据定位到的行与列的交叉点元素中的权限集合判定是否拥有所需的权限。

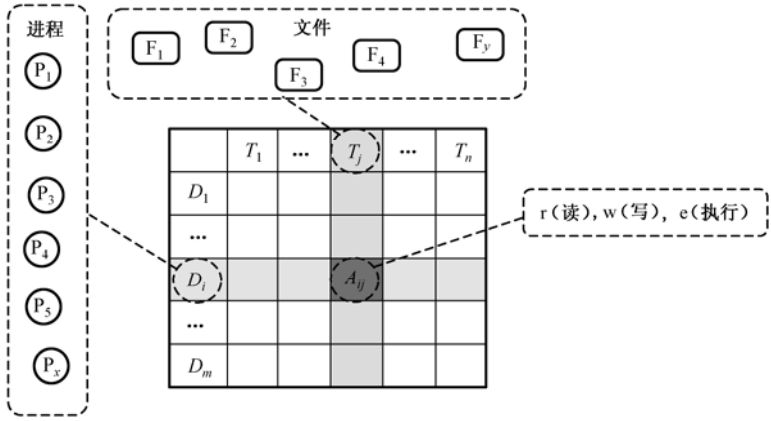


图 5.1　TE 模型的域定义表（DDT）

例 5.1　在图 5.1 所示的域定义表中，设进程 P_x 欲读文件 F_y，请简要说明判定进程 P_x 是否能读文件 F_y 的过程。

解答：根据域标签确定与进程 P_x 对应的域 D_i，根据类型标签确定与文件 F_y 对应的类型 T_j，在 DDT 表中找到 D_i 行与 T_j 列的交叉元素 A_{ij}，如果 A_{ij} 中含有读权限，则允许进程 P_x 读文件 F_y，否则不允许进程 P_x 读文件 F_y。

主体有时可以成为客体。例如，一般情况下，进程是主体，当一个进程向另一个进程发信号（Signal）时，接收信号的进程则是客体。TE 模型在 DDT 表中不描述这样的客体。

只有在主体发生相互作用时，主体才转变为客体。TE 模型用另一张表来实现主体与主体之间进行相互作用时的访问控制，该表称为域相互作用表（DIT，Domain Interaction Table）。DIT 表的行和列都与域相对应，行与列交叉点的元素表示行中的域对列中的域的访问权限，访问权限包括发信号、创建进程、杀死进程等。

例 5.2　设操作系统中的进程 P_x 欲向进程 P_y 发信号，请简要说明在 TE 模型的访问控制框架下判定进程 P_x 是否可以向进程 P_y 发信号的过程。

解答：根据域标签确定与进程 P_x 对应的域 D_i 及与进程 P_y 对应的域 D_j，在 DIT 表中找到 D_i 行与 D_j 列交叉点的元素 A_{ij}，如果 A_{ij} 中含有“发信号”（Signal）权限，则允许进

程 P_x 向进程 P_y 发信号，否则不允许进程 P_x 向进程 P_y 发信号。

TE 模型根据主体的域标签、客体的类型标签、DDT 表中的授权或 DIT 表中的授权进行访问控制，只有系统管理员或系统安全管理员才能确定域标签、类型标签、DDT 表中的授权和 DIT 表中的授权，所以，TE 模型可以实现强制访问控制。

定理 5.1　TE 模型是一个强制访问控制模型。

TE 模型根据主体的域属性控制主体的访问行为，而不根据主体的用户标识来进行访问控制，不管是什么用户，都只能在所在域的访问权限范围内工作。UNIX 操作系统中的 root 超级用户的行为也因此受到约束，从而失去无所不能的特权。

可以把 TE 模型的 DDT 表看成是一个粗粒度的访问控制矩阵。访问控制矩阵的每一行对应一个主体，而 DDT 表的每一行对应一组主体。访问控制矩阵的每一列对应一个客体，而 DDT 表的每一列对应一组客体。

控制一组主体对一组客体的访问行为，从某种意义上说，可以反映应用系统的实际访问情况。这是因为，一个应用系统通常包含一组进程，它们通常需要访问一组文件。

通过域的划分和类型的划分，并建立域与类型之间的对应关系，可以把主体对客体的访问限定在确定的范围之内，从而实现应用系统的有效隔离。

例 5.3　设在一个操作系统中需要运行 Web、E-mail、FTP 等多种应用系统，请采用 TE 模型的访问控制方法，给出一个访问控制方案，要求使各种应用系统能够互不干扰地进行工作。

解答：为 Web、E-mail、FTP 等应用各定义一个域，TE 模型实现在的应用隔离如图 5.2 所示，域的名称分别设为 web_p_d、mail_p_d、ftp_p_d 等，为各种应用各定义一个类型，类型的名称分别设为 web_f_t、mail_f_t、ftp_f_t 等。

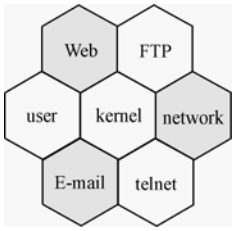


图 5.2　TE 模型实现的应用隔离

以 Web 应用为例，把该应用的所有专用文件归入 web_f_t 类型中，把该应用的所有进程归入 web_p_d 域中，根据实际需要，在 DDT 表中定义域 web_p_d 对类型 web_f_t 的访问权限。这样，只有域 web_p_d 中的进程才能访问类型 web_f_t 中的文件。

类似地，可以在 DDT 表中为 E-mail 和 FTP 等应用定义相应的访问权限。这样，可以使各个应用中的进程能够正常地访问各自的文件、正常地工作，但是，它们不能访问别的应用中的文件，从而实现不同应用之间互不干扰的工作。

以上简化的例子说明了通过 TE 模型的访问控制方法实现应用系统隔离的基本思想。通过应用域的划分，能够为应用系统建立相对独立的运行空间，使一个应用系统影响不到其他应用系统的工作。

例如，FTP 应用系统的工作影响不到 Web 应用系统的工作，更影响不到操作系统内核的工作。在系统被攻破的情况下，攻入 FTP 应用系统的非法用户只能在该应用系统的范围内活动，难以对系统的其他部分进行访问，从而缩小系统受侵害的范围。

虽然 TE 模型灵活性好且功能强大，但在实际应用中存在以下问题。

（1）访问控制权限的配置比较复杂：当系统中应用较多、进程较多、文件数较大时，配置 DDT 表和 DIT 表、为每个进程分配域、为每个文件分配类型等涉及大量的工作。

（2）二维表结构无法反映系统的内在结构：文件系统中目录与子目录的关系表现出客体间存在层次结构，进程中的父进程与子进程的关系表现出主体间也存在层次结构，DDT 表和 DIT 表无法反映这种结构。

（3）控制策略的定义需要从零开始：TE 模型只提供了访问控制框架，没有提供访问控制规则，什么样的域可以对什么样的类型进行何种访问，TE 模型没有解答这样的问题，系统管理员需要为每个应用设计相应的访问控制内容。

5.1.2　DTE模型的基本思想

DTE 模型是 TE 模型的改进版本，它立足于解决 TE 模型在实际应用中遇到的问题。

定义 5.3　DTE 模型用于描述安全属性和访问控制配置的高级语言形式的策略描述语言称为 DTE 语言（DTEL，DTE Language）。

与 TE 模型相比，DTE 模型具有以下两个突出的特点。

（1）使用高级语言描述访问控制策略：提供 DTEL，用于取代 TE 模型的二维表，描述安全属性和访问控制配置。

（2）采用隐含方式表示文件安全属性：在系统运行期间，利用内在的客体层次结构简明地表示文件的安全属性，以摆脱对存储在物理介质上的文件属性的依赖。

DTE 模型的 DTEL 安全策略描述语言提供的主要功能包括类型描述、类型赋值、域描述和初始域设定等方面。通过 DTEL 提供的主要功能，可以了解 DTE 模型访问控制的基本思想。

DTEL 的类型描述功能定义 DTE 模型中的客体类型。

例 5.4　以下是 DTEL 的类型描述语句，请说明其含义。

```
type unix_t,specs_t,budget_t,rates_t;
```

解答：该语句定义了 DTE 模型中的 4 个客体类型，类型名称分别是 unix_t、specs_t、budget_t 和 rates_t。

DTEL 的类型赋值功能把客体与客体类型关联起来，也就是为客体设定类型属性，或者说，把类型标签值赋给客体。一个客体只能对应一个类型。

利用客体间的层次结构关系，可以采用隐含赋值的方法给客体赋类型值，即给文件系统中的一个目录赋类型值，相当于把该类型值赋给该目录及其下的所有子目录和文件，除非给该目录下的子目录或文件另外赋类型值。也就是说，DTE 模型实施以下的客体赋值的隐含规则。

规则 5.1　如果没有显式的给文件系统中的一个客体赋类型值，那么，该客体的类型值与其父目录的类型值相同。

规则 5.1 的实施属于递归赋值，在 DTEL 的类型赋值语句中可以用“-r”选项来注明。在类型赋值语句中，还可以用“-s”选项来禁止系统在运行期间创建与目录的类型不同的客体。

例 5.5　以下是 DTEL 的类型赋值语句，请说明其含义。

```
assign -r -s unix_t      /;
assign -r -s specs_t     /subd/specs;
assign -r -s budget_t    /subd/budget;
assign -r -s rates_t     /subd/rates;
```

解答：这些赋值语句分别把目录/、/subd/specs、/subd/budget、/subd/rates 的类型设定为 unix_t、specs_t、budget_t、rates_t。

由于把根目录/ 的类型设定为 unix_t，所以，隐含地，文件系统中所有客体的类型都是 unix_t，但是，/subd/specs、/subd/budget、/subd/rates 这三个目录下的客体除外，因为它们有显式设定的类型。

“-r”选项表示实施递归赋值。以第二个语句为例，“-s”选项表示禁止系统在运行期间在目录/subd/specs 下创建类型值不等于 specs_t 的客体。

DTEL 的域描述功能除了定义 DTE 模型中的主体域以外，还定义域的入口点（Entry Point），并设定域对类型的访问权限，以及域对其他域的访问权限。

一个域的一个入口点是一个可执行程序，执行该可执行程序可以使执行它的进程进入到该域中。设定对类型的访问权限时，用字符 r、w、x、d 分别表示读、写、执行、搜索（目录）权限。设定对域的访问权限时，用关键字 exec 和 auto 表示执行入口点程序的权限。当域 *A* 拥有对域 *B* 的 exec 权限时，域 *A* 中的主体 *S* 可以执行域 *B* 中的入口点程序 P_b，并可以执行域切换操作进入域 *B*，使执行域 *B* 的入口点程序 P_b 的进程在域 *B* 中运行。当域 *A* 拥有对域 *C* 的 auto 权限时，域 *A* 中的主体 *S* 可以执行域 *C* 中的入口点程序 P_c，并自动进入域 *C*，使执行域 *C* 的入口点程序 P_c 的进程在域 *C* 中运行。

例 5.6　以下是 DTEL 的域描述语句，请说明其含义。

```
#define DEF (/bin/sh),(/bin/csh),(rxd->unix_t)
domain engineer_d     = DEF,(rwd->specs_t);
```

```
domain project_d      = DEF,(rwd->budget_t),(rd->rates_t);
domain accounting_d  = DEF,(rd->budget_t),(rwd->rates_t);
```

解答：第一个语句定义了一个宏 DEF，后三个语句定义了 engineer_d、project_d、accounting_d 三个域，这三个域都包含/bin/sh 和/bin/csh 两个入口点程序，都对 unix_t 类型拥有读、执行、搜索权限，另外，各域还分别拥有如下属性。

engineer_d 域：对 specs_t 类型拥有读、写、搜索权限；

project_d 域：对 budget_t 类型拥有读、写、搜索权限，对 rates_t 类型拥有读、搜索权限；

accounting_d 域：对 budget_t 类型拥有读、搜索权限，对 rates_t 类型拥有读、写、搜索权限。

DTEL 的初始域设定功能设定操作系统中第一个进程的工作域。操作系统中的第一个进程是所有进程的祖先，所有进程都由它派生出来。在进程派生过程中，子进程继承父进程的工作域，即子进程在父进程所在的域中运行。当进程的工作域拥有对另一个域的 exec 或 auto 权限时，进程可以执行另一个域的入口点程序，并使另一个域成为自己的工作域。可见，只要确定了第一个进程的工作域，就有办法确定所有进程的工作域。

例 5.7　以下是 DTEL 的系统域描述和初始域设定语句，请说明其含义。

```
domain system_d      = (/etc/init),(rwxd->unix_t),(auto->login_d);
domain login_d  = (/bin/login),(rwxd->unix_t),
                        (exec->engineer_d,project_d,accounting_d);
initial_domain   = system_d;
```

解答：前两个语句定义了 system_d 和 login_d 两个系统域，第三个语句把 system_d 设定为操作系统第一个进程的工作域，即操作系统的第一个进程将在该域中运行。system_d 域包含/etc/init 这个入口点程序，对 unix_t 类型拥有读、写、执行、搜索权限，对 login_d 域拥有 auto 权限。login_d 域包含/bin/login 这个入口点程序，对 unix_t 类型拥有读、写、执行、搜索权限，对 engineer_d、project_d 和 accounting_d 域拥有 exec 权限。

UNIX 操作系统的第一个进程执行/etc/init 程序，该进程称为 init 进程。根据例 5.7 的配置，init 进程在 system_d 域中运行，init 进程创建的子进程 P_{child} 也在 system_d 域中运行。P_{child} 进程可以执行 login_d 域的/bin/login 入口点程序，P_{child} 进程执行/bin/login 程序后称为 P_{login} 进程，P_{login} 进程自动进入 login_d 域中运行。P_{login} 进程可以执行 engineer_d、project_d 和 accounting_d 域的入口点程序，并可以执行域切换操作进入到相应的域中运行。

DTE 模型通过 DTEL 描述访问控制策略的配置从而实现访问控制，DTEL 类型描述

功能定义类型，类型赋值功能把类型值赋给客体，域描述功能定义域、域权限、域与主体的关系及主体的域切换（Domain Transition）方法，初始域设定功能设定第一个进程的工作域，从而为整个系统中的进程的工作域的确立基础。

推论 5.1　DTE 模型是一个强制访问控制模型。

5.2　SELinux实现的TE模型

SELinux 是在美国国家安全局（NSA，National Security Agency）的支持下开发出的安全增强型 Linux 系统（Security Enhanced Linux），是一个开放源代码的操作系统安全增强系统。SELinux 实现的访问控制模型的核心是 DTE 模型，但该系统称其为 TE 模型，为明确起见，本章把它称为 SETE（SELinux Type Enforcement）模型，该模型提供了专门的安全策略配置语言，本章把它称为 SEPL（SELinux Policy Language）。SEPL 用于配置 SETE 模型的访问控制策略，其作用与 DTE 模型的 DTEL 相仿。

5.2.1　SETE模型与DTE模型的区别

SETE 模型对 DTE 模型进行了扩充和发展，与 DTE 模型相比，SETE 模型具有以下突出的特点。

（1）类型的细分：DTE 模型把客体划分为类型，针对类型确定访问权限，SETE 模型在类型概念的基础上增加客体的类别（Class）概念，针对类型和类别确定访问权限；

（2）权限的细化：SETE 模型为客体定义了几十个类别，为每个类别定义了相应的访问权限，因此，模型中定义了大量精细的访问权限。

定义 5.4　一般情况下，SETE 模型把“域”和“类型”都统称为“类型”，在需要明确区分之处，它把“域”称为“域类型”或“主体类型”。

例 5.8　请列出 SETE 模型中几个常用的客体类别，并举出它们的几个常见权限。

解答：file（普通文件）、dir（目录）、process（进程）、socket（套接字）和 filesystem（文件系统）等都是 SETE 模型中的常用客体类别。

file 类别的常见权限有 read（读）、write（写）、execute（执行）、getattr（取属性）、create（创建）等。dir 类别的常见权限有 read（读）、write（写）、search（搜索）、rmdir（删除）等。process 类别的常见权限有 signal（发信号）、transition（域切换）、fork（创建子进程）、getattr（取属性）等。socket 类别的常见权限有 bind（绑定名字）、listen（侦听连接）、connect（发起连接）、accept（接受连接）等。filesystem 类别的常见权限有 mount（安装）、unmount（卸载）等。

SETE 模型定义了几十个类型，例 5.8 仅列出了其中的几个，各个类型的访问权限非常丰富，例 5.8 也只举出了其中的几个，由此即可见 SETE 模型访问权限的粒度细化程度。

推论 5.2　SETE 模型是一个强制访问控制模型。

5.2.2　SETE模型的访问控制方法

在 SELinux 中，所有的访问都必须明确授权，在默认情况下，所有的访问都是不允许的，只有经过授权的访问才是允许的。SETE 模型通过 SEPL 描述访问控制策略，确定访问控制的授权方法。

SEPL 的 allow 规则是描述访问控制授权的基本方法，allow 规则包含以下四个元素。

- 源类型（source_type）：主体的域，即域类型或主体类型，主体通常是要实施访问操作的进程；
- 目标类型（target_type）：由主体访问的客体的类型；
- 客体类别（object_class）：访问权限所针对的客体类别；
- 访问权限（perm_list）：允许源类型对目标类型的客体类别进行的访问。

allow 规则的一般形式是：

```
allow source_type target_type : object_class perm_list;
```

例 5.9　以下是 SEPL 的访问授权规则，请说明其含义。

```
allow user_d bin_t : file {read execute getattr};
```

解答：该规则把对 bin_t 类型的 file 类别的客体的 read、execute 和 getattr 访问权限授给 user_d 域的主体，允许 user_d 域的进程对 bin_t 类型的普通文件进行读、执行和取属性的操作，取属性就是查看文件的属性信息，如日期、时间、属主等。

在例 5.9 中，假设 user_d 域包含的是普通用户进程，如登录进程，bin_t 类型包含的是可执行程序文件，如/bin/bash 命令解释程序，则该规则授权普通用户的登录进程执行 bash 命令解释程序。

例 5.10　在 Linux 操作系统中，/etc/shadow 文件保存用户的口令信息，passwd 程序管理口令信息，为用户提供修改口令的功能，设两个文件在 Linux 中的部分权限信息如下：

```
r--------    root  root  ..... shadow
r-s--x--x    root  root  ..... passwd
```

请说明 passwd 程序为普通用户修改口令的方法，该方法存在什么不足？如何利用 SETE 模型的访问控制克服该不足？

解答：口令信息存放在 shadow 文件中，用户修改口令时，必须修改该文件的内

容，但普通用户没有访问该文件的权限。用户执行 passwd 程序修改口令，该程序文件的 SETUID 控制位是打开的（由权限中的字符 s 表示），用户进程执行该程序时，进程的有效身份变成 root 用户，由于 root 用户是 shadow 文件的属主，所以，具有 root 用户身份的进程可以访问 shadow 文件，从而为用户修改口令信息。

passwd 程序修改口令的方法是采用 SETUID 机制，使执行该程序的用户进程的有效身份变为 root 用户，目的是使用户进程能够修改 shadow 文件中的口令信息。由于任何用户都能执行 passwd 程序，所以该方法实际上使任何用户的进程都能拥有 root 用户的权限，但是，在 Linux 系统中，root 用户不仅能够访问并修改 shadow 文件中的口令信息，它具有无所不能的特权，无形中，该方法使任何用户的进程都能具有无所不能的特权，这是一种潜在的巨大危险。

利用 SETE 模型，可以定义一个包含 passwd 进程的 passwd_d 域，定义一个包含 shadow 文件的 shadow_t 类型，配置以下规则授权 passwd_d 域的进程访问 shadow_t 类型的文件：

```
allow passwd_d shadow_t : file {ioctl read write create getattr
    setattr lock relabelfrom relabelto append unlink link rename};
```

这个规则给 passwd_d 域中的 passwd 进程授予修改 shadow_t 类型的 shadow 文件中的口令信息所需要的访问权限。这个规则使 passwd 进程拥有访问 shadow_t 类型的文件的权限，但不拥有其他的权限，从而克服了以上方法的不足。

Linux 系统修改 shadow 文件中的口令信息的方法是首先移动该文件，然后创建一个新的 shadow 文件，例 5.10 中的授权规则提供了执行这些操作所需要的各种权限。

SELinux 系统的访问控制是在 Linux 系统的访问控制的基础上增加的访问控制，一个操作要想在 SELinux 系统中得到允许，首先必须在 Linux 系统中得到允许，所以，在例 5.10 中，利用 SETE 模型的规则对 passwd 进程进行访问控制是以 SETUID 机制为基础的。例 5.10 可以用图 5.3 加以描述。

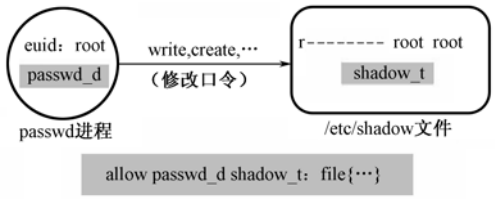


图 5.3　用 SETE 模型控制 passwd 进程

5.2.3　授权进程切换工作域

为了确保进程的行为不威胁系统的安全性，需要确保进程在正确的域中运行正确的程序。例如，针对例 5.10 给出的修改口令的情况，我们不希望在 passwd_d 域中运行的

进程运行不应该访问 shadow 文件的程序。换句话说，我们必须使运行指定程序的进程在合适的域中运行。问题是应该如何选择和设定进程运行时应该进入的域？为了弄清这些问题，首先需要掌握用户在系统中执行操作时涉及的进程的工作过程。

例 5.11　设用户 BOB 登录进入 SELinux 系统后欲修改其口令，试分析与该过程有关的进程可能涉及的域的情况，以及可能遇到的访问权限问题。

解答：可以用图 5.4 表示用户 BOB 登录系统后修改口令的过程。设普通用户进程在 user_d 域中运行，用户登录后运行 bash 进程，则该进程在 user_d 域中运行。口令文件 shadow 的类型是 shadow_t，user_d 域无权访问该类型的文件。负责修改口令的 passwd 进程在 passwd_d 域中运行，该域可以访问 shadow_t 类型的 shadow 文件。用户在 bash 进程中执行 passwd 程序可以生成 passwd 进程。所遇到的问题是在 user_d 域中生成的 passwd 进程如何进入 passwd_d 域。

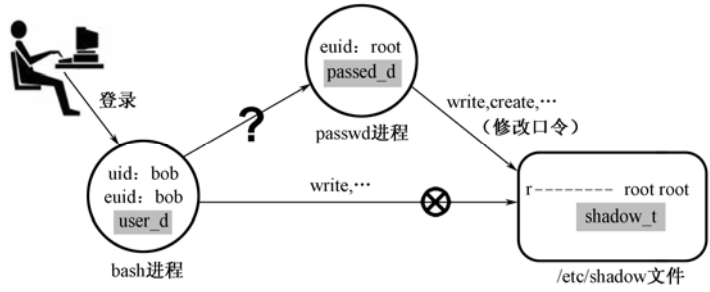


图 5.4　用户登录后欲修改口令的过程

在例 5.11 中，由于用户运行的 bash 进程的工作域 user_d 没有访问 shadow_t 类型的 shadow 口令文件的权限，所以需要想办法使在 user_d 域中生成的 passwd 进程进入到一个有权限访问 shadow 口令文件的域中运行。

在标准 Linux 系统中也存在类似的情况，在那里，用户运行的 bash 进程的有效身份没有访问 shadow 口令文件的权限，需要设法使由该进程生成的 passwd 进程拥有一个有权限访问 shadow 口令文件的有效身份。解决该问题采用的是 SETUID 的方法。

标准 Linux 系统改变 passwd 进程的有效身份的目的是要使该进程获得访问 shadow 口令文件的权限，SETE 模型欲切换 passwd 进程的工作域的目的也是要使该进程获得访问 shadow 口令文件的权限。

例 5.10 曾讨论过标准 Linux 系统改变 passwd 进程的有效身份的方法，这里，对其过程做进一步分析，看看能否从中借鉴有意义的思想，以便设计出切换 passwd 进程的工作域的方法。

例 5.12　设用户 BOB 登录进入 Linux 系统后欲修改其口令，试分析该过程通过改变进程的有效身份以获得访问 shadow 口令文件的权限的方法。

解答：可以用图 5.5 表示用户 BOB 登录系统后修改口令的过程。设用户 BOB 登录后运行 bash 进程，则该进程的真实身份和有效身份都是 bob，由 shadow 文件的权限位知，除 root 用户以外的用户都没有权限访问该文件。用户在 bash 进程中执行 passwd 程序时，bash 进程首先为此通过 fork 系统调用创建一个子进程，不妨记为 bash_c 进程，随后，bash_c 进程通过 exec 系统调用执行 passwd 程序。bash_c 进程的真实身份和有效身份与其父进程 bash 的相同，都是 bob，显然，它无法访问 shadow 文件。

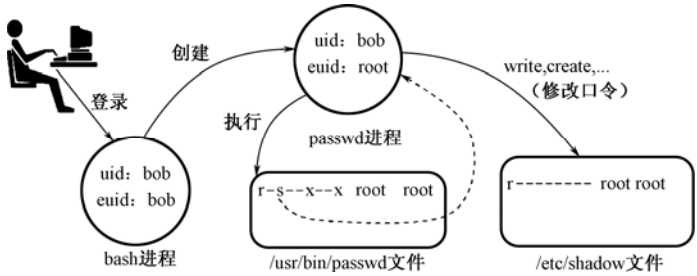


图 5.5　进程在 Linux 中改变有效身份后修改口令

为解决这个问题，系统设计人员或管理人员打开了 passwd 程序文件的 SETUID 控制位，在这种情况下，当 bash_c 进程通过 exec 系统调用执行 passwd 程序时，bash_c 进程的有效身份被设置为 passwd 文件的属主身份，那就是 root 用户。程序映像被 passwd 程序替代后的 bash_c 进程就是 passwd 进程，所以，passwd 进程的有效身份是 root 用户，因而，它获得了访问 shadow 口令文件的权限。

例 5.12 中的分析表明，标准 Linux 系统改变进程有效身份的方法有两个关键点：一是打开要执行的 passwd 可执行程序文件中的 SETUID 控制位；二是当进程通过 exec 系统调用更换程序映像时把进程的有效身份设置成可执行程序文件的属主身份。

打开的被执行的可执行程序文件中的 SETUID 控制位既是一种授权，也是触发进程有效身份变更的一个控制点。我们不妨也从被执行的可执行程序文件入手，寻求进程运行时的域切换的授权与触发问题的解决方案。

例 5.11 分析了用户 BOB 登录后想要修改口令的过程，现在要完成的任务是确定一个方案，使在 user_d 中生成的 passwd 进程能够进入 passwd_d 域中运行。

例 5.13　设用户 BOB 登录进入 SELinux 系统后欲修改其口令，已知 shadow 口令文件是 shadow_t 类型的文件，passwd_d 域拥有修改 shadow_t 类型的口令文件所需要的访问权限，试给出一个确定进程工作域的方案，使负责口令修改的 passwd 进程有权修改 shadow 文件中的口令信息。

解答：用图 5.6 表示用户 BOB 登录系统后修改口令的过程。用户 BOB 登录后运行的 bash 进程在 user_d 域中运行，该域无权访问 shadow_t 类型的文件。用户在 bash 进程中执行 passwd 程序时，bash 进程通过 fork 系统调用创建一个子进程，不妨记为 bash_c

进程，随后，bash_c 进程通过 exec 系统调用执行 passwd 程序。

passwd 程序是 passwd_exec_t 类型的文件，为使在 user_d 域中运行的 bash_c 进程能够执行 passwd 程序，需要授权 user_d 域执行 passwd_exec_t 类型的文件，参见图 5.6 中第一个 allow 规则。

exec 系统调用执行后，bash_c 进程的程序映像被 passwd 程序替代，成为 passwd 进程。在正常情况下，这样得到的 passwd 进程仍在 user_d 域中运行。为使 passwd 进程能够进入 passwd_d 域运行，需要给 user_d 域中的进程授予进入 passwd_d 域的权限，即授权进程从 user_d 域切换成 passwd_d 域，参见图 5.6 中第三个 allow 规则。

bash_c 进程是在把程序映像切换成 passwd 程序后变成 passwd 进程的，此刻是使进程进入 passwd_d 域的最佳时机，因此，exec 系统调用装入 passwd 程序的操作可以作为进程域切换的触发点，这需要把 passwd 程序定义为 passwd_d 域的入口点，同时授权 passwd_d 域把 passwd 程序作为入口点，可行的做法是给 passwd_d 域授予把 passwd_exec_t 类型的文件作为入口点的权限，图 5.6 中第二个 allow 规则完成这项任务。

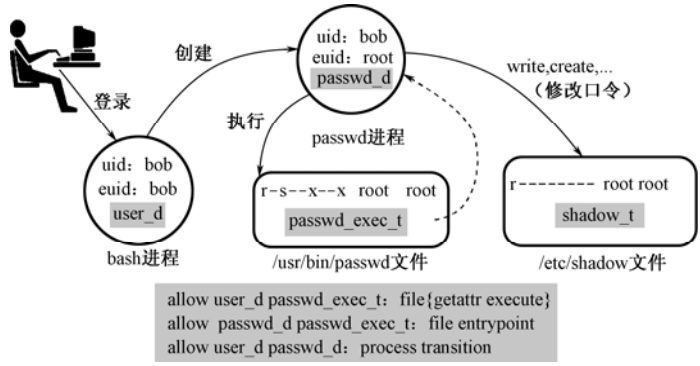


图 5.6　进程在 SELinux 中切换工作域后修改口令

例 5.13 给出的方案主要由以下三个方面的工作组成。

- 给 user_d 域授予执行 passwd_exec_t 类型的文件的权限，这归结为如下规则：

 allow user_d passwd_exec_t: file {getattr execute};

- 给 passwd_d 域授予把 passwd_exec_t 类型的文件作为入口点的权限，这归结为如下规则：

 allow passwd_d passwd_exec_t: file entrypoint;

- 给进程的 user_d 域授予切换成 passwd_d 域的权限，这归结为如下规则：

 allow user_d passwd_d: process transition;

根据以上例子给出的方案，可以总结出一般化的结论，即以上三个规则共同构成了进程切换工作域的条件。需要注意的是：三个规则必须同时具备，缺一不可。也就是说，进程切换域的条件如下。

条件 5.1　进程要实现从旧的工作域到新的工作域的切换，必须同时满足以下三个

条件：

（1）进程的新的工作域必须拥有对可执行文件的类型的 entrypoint 访问权限；

（2）进程的旧的工作域必须拥有对入口点程序的类型的 execute 访问权限；

（3）进程的旧的工作域必须拥有对进程的新的工作域的 transition 访问权限。

5.2.4　进程工作域的自动切换

一个进程满足条件 5.1 的要求表示该进程拥有了从一个域切换到另一个域的条件，但并不表示域切换事件一定发生，要实现域的切换还必须执行域切换操作。

回顾 DTE 模型，它提供了与域切换相关的 exec 权限和 auto 权限。如果域 *A* 拥有对域 *B* 的 exec 权限或 auto 权限，那么，域 *A* 中的进程 P 可以通过 exec 系统调用执行域 *B* 中的入口点程序 F_b。当域 *A* 拥有的是 exec 权限时，如果进程 P 要求进入域 *B*，那么，exec 系统调用执行程序 F_b 后，进程 P 从域 *A* 切换到域 *B*，如果进程 P 不要求进入域 *B*，那么，exec 系统调用执行程序 F_b 后，进程 P 不会切换域 *B*。当域 *A* 拥有的是 auto 权限时，那么，exec 系统调用执行程序 F_b 后，进程 P 自动从域 *A* 切换到域 *B*。

SETE 模型也采取类似的域切换方法，由 exec 系统调用触发域切换操作，支持自动切换和按要求切换。按要求切换就是仅当进程要求切换时才进行切换，如果进程不提出要求，就不进行切换。自动切换则不同，无须进程关心切换的问题，只要 exec 系统调用执行入口点程序，就进行域的切换。

SETE 模型通过类型切换（Type Transition）规则描述进程工作域的自动切换方法，该规则的形式是：

type_transition source_type target_type : process default_type;

其中，source_type、target_type 和 default_type 分别表示进程的当前域、入口点程序文件的类型和进程的默认域。

该规则所确定的指令是：当在 source_type 域中运行的进程通过 exec 系统调用执行 target_type 类型的入口点程序时，系统自动尝试把进程的工作域切换为 default_type 域。域切换的尝试是否成功取决于条件 5.11 的要求是否得到满足。

例 5.14　假设为了使用户 BOB 登录进入 SELinux 系统后能够修改其口令，已按照例 5.13 的方案进行了域切换的授权，试给出实现域的自动切换的规则。

解答：实现域的自动切换的规则是：

type_transition user_d passwd_exec_t : process passwd_d;

在该规则的控制下，当在 user_d 域中运行的 bash_c 进程通过 exec 系统调用执行 passwd_exec_t 类型的入口点程序 passwd 时，系统将尝试自动把 bash_c 进程的工作域切换为 passwd_d 域，由于例 5.13 的授权，是满足的条件 5.11 的要求的，所以，域切换尝试可以成功。此时的 bash_c 进程就是 passwd 进程，因此，系统自动使 passwd 进程进入

到 passwd_d 域中运行。

一般而言，用户都不希望为诸如切换进程的工作域这样的事情而烦心，他们只关心手中要完成的工作。例如，用户 BOB 运行 passwd 程序的目的是修改口令，他希望系统能够按照他的意愿完成口令的修改任务，别的事情并不是他想关心的。

通过使用 type_transition 规则，SETE 模型允许访问控制策略配置人员指示系统在不需要用户参与的情况下自动为进程完成域的切换工作。

5.3　访问判定与切换判定

SELinux 系统提供两种基本的判定：访问判定（Access Decision）和切换判定（Transition Dicision）。

定义 5.5　在 SELinux 系统中，为给定的主体能否在给定的客体上实施给定的操作做出结论的过程称为访问判定。

定义 5.6　在 SELinux 系统中，为是否需要切换新创建的进程和文件的类型标签做出结论的过程称为切换判定，切换判定也称为标记判定（Labeling Decision）。

在定义 5.6 中，对于进程而言，类型指的就是域。本节简要介绍 SELinux 系统实现的访问判定和切换判定的基本方法。

5.3.1　SELinux的访问判定

访问判定是对访问请求的响应，访问判定以访问控制策略为依据，访问判定的基本思想是检查是否存在访问控制规则对请求的访问进行过授权，判定的结果是访问控制策略反映的有关请求的访问操作的授权结论。

根据 5.2.2 节的介绍，SELinux 系统的 SEPL 用源类型（source_type）、目标类型（target_type）、客体类别（object_class）和访问权限（perm_list）四个元素描述访问控制授权，因此，从概念上说，可以通过以下的四元组来描述一个访问请求：

(source_type, target_type, object_class, perm_list)

可以针对 perm_list 描述判定的结果。与 perm_list 相对应，可以用位图来表示判定结果，位图中的每一位表示一个访问权限，可以用 1 表示授权，用 0 表示不授权。SELinux 系统把表示判定结果的位图称为访问向量（AV，Access Vector），因为 perm_list 是与客体类别对应的，所以 AV 也与客体类别对应。

例 5.15　试举例说明用于表示 SELinux 系统的访问判定结果的访问向量的基本形式。

解答：针对每种客体类别设计一种访问向量（AV），与 file（文件）类别相对应的

file（文件）类别的访问权限										
append	create	execute	get attribute	I/O control	link	lock	read	rename	unlink	write
?	?	?	?	?	?	?	?	?	?	?

图 5.7　file（文件）类别的简化访问向量

AV 可以用如图 5.7 所示的简化形式表示。文件类别的访问权限包括 create、read、write、execute 等，图 5.7 中的简化示例只列出了其中的一小部分，实际的 AV 列出该客体类别的所有访问权限，客体类别共有多少种访问权限，该客体类别的 AV 就有多少位，每种访问权限对应 AV 中的一位。图中的“？”要么是 1，表示授予对应的权限；要么是 0，表示没有授予对应的权限。

例 5.15 描述了 AV 的基本形式及其内容的含义。在 SELinux 系统实现的 SETE 模型的访问机制中，对于每一个访问请求，如果能在访问控制策略中找到和它匹配的访问控制规则，则访问判定返回的结果包含三个相关联的 AV，分别是 allow 型 AV、auditallow 型 AV 和 dontaudit 型 AV。

allow 型 AV 主要描述那些允许主体在客体上实施的操作，除非 auditallow 型 AV 中有特别说明，否则，这些操作的实施无须进行审计。auditallow 型 AV 主要描述那些在实施时需要审计的操作。dontaudit 型 AV 主要描述那些不允许主体在客体上实施的且不需要进行审计的操作，即不必记录该操作请求遭到拒绝。

例 5.16　设 SELinux 系统中的进程请求在文件上实施操作，试举例说明访问判定返回结果的基本构成。

解答：以例 5.15 列出的 file（文件）类别的访问权限为例，访问判定返回的简化的访问向量如图 5.8 所示。该结果的 allow 型 AV 表示允许进程在文件上实施 append（附加内容）和 create（创建文件）操作，因为与这些操作对应的 auditallow 型 AV 的值为 0，所以，进程在文件上实施的 append 或 create 操作是不需要进行审计的。如果与 append 或 create 操作对应的 auditallow 型 AV 的值为 1，则系统要对进程在文件上实施的该操作进行审计。

	文件类别的访问权限										
	append	create	execute	get attribute	I/O control	link	lock	read	rename	unlink	write
allow	1	1	0	0	0	0	0	0	0	0	0
auditallow	0	0	0	0	0	0	0	0	0	0	0
dontaudit	0	0	0	0	0	0	0	0	0	0	0

图 5.8　访问判定返回的简化的访问向量

假如进程在访问请求中申请在文件上实施 write 操作，该操作在 allow 型 AV 中的对

应值为 0，这表明系统拒绝了该请求，由于 dontaudit 型 AV 中与该操作对应的值为 0，因而，系统将审计“write 操作请求被拒绝”的事件，但如果 dontaudit 型 AV 中与该操作对应的值为 1，则系统将不审计该事件。

例 5.16 描述了文件类别的访问判定返回结果的基本构成，并具体说明了 allow 型 AV、auditallow 型 AV 和 dontaudit 型 AV 的实际含义。结合例 5.16 表达的意义，可以进一步总结出 SELinux 系统进行访问判定的原则如下。

（1）除非在访问控制策略中有匹配的访问控制规则明确授权主体在客体上实施指定的操作，否则操作申请一概被拒绝。

（2）一旦操作申请被拒绝，系统将审计该操作被拒绝的事件，除非系统明确说明无须对该事件进行审计。

（3）如果系统对已授权的某操作有明确的审计要求，则当主体实施该操作时，系统对该操作进行审计。

原则（1）意味着在访问控制策略中没有访问控制规则与其匹配的访问请求必然被拒绝，allow 型 AV 描述没有被该原则拒绝的操作。dontaudit 型 AV 描述原则（2）中无须审计的操作被拒绝事件。auditallow 型 AV 描述原则（3）中需要审计的操作。

SELinux 系统根据以上访问判定原则进行判定所得到的结果情况可以简要概括为如表 5-1 所示的形式。

表 5-1　访问判定结果概要

判 定 结 果	是否授权访问	是否进行审计
在访问控制规则中没有匹配	不授权	审计判定结果
allow 型 AV 值是 1	授权	一般不审计
auditallow 型 AV 值是 1	不表示授权	审计访问
dontaudit 型 AV 值是 1	不表示授权	不审计判定结果

在 SELinux 系统中，主体对客体进行操作前，需要根据 AV 检查操作是否可以实施。出于提高系统运行效率的考虑，为了避免每次操作前都必须进行一次原始的访问判定，系统提供对 AV 的缓存信息。缓存 AV 的数据结构称为访问向量缓存（AVC，Access Vector Cache）。

在提供 AVC 支持的情况下，当主体要对客体进行操作时，系统首先从 AVC 中查找与访问请求相符的 AV，如果找到，就根据该 AV 确定是否允许，可以节省根据访问控制策略构造 AV 的时间开销，只有在 AVC 中找不到相符的 AV 时，才需要从头进行原始的访问判定。

5.3.2　SELinux的切换判定

SETE 模型以主体的域和客体的类型为依据进行访问控制，前面着重介绍了访问控制的基本方法，现在讨论给主体分配域标签和给客体分配类型标签的方法。

与 DTE 模型一样，SETE 模型在确定主体的域和客体的类型时充分考虑了主体的层次结构和客体的层次结构。DTE 模型和 SETE 模型反映的主体的层次结构主要是父进程和子进程之间的关系构成的层次结构。反映的客体的层次结构主要是父目录、子目录、文件之间的关系构成的层次结构。

一般情况下，创建新进程时，将父进程的域标签作为新进程的域标签，创建新文件或新目录时，将父目录的类型标签作为新文件或新目录的类型标签。但有时，需要给新的主体或新的客体分配新的标签，切换判定就是要确定是否需要给新的主体或新的客体分配新的标签及新的标签应该取什么值。给主体或客体分配新的标签就称为标签切换。

例 5.17　试举例说明在 SELinux 系统中创建新进程时不切换域标签和切换域标签的情形。

解答：进程创建与域切换如图 5.9 所示，设执行 vi 文本编辑程序的 vi 进程在 vi_d 域中运行，在 vi 进程中，可以执行 ls 命令查看文件和目录的描述信息，执行 ls 命令的 ls 进程也在 vi_d 域中运行，没有发生域标签切换，即 ls 进程的域标签等于它的父进程（vi 进程）的域标签（vi_d）。

设系统的 init 进程在 initrc_d 域中运行，init 进程为安全 shell 服务创建的 ssh 守护进程将在 sshd_d 域中运行，ssh 守护进程的域标签（sshd_d）不等于它的父进程（init 进程）的域标签（initrc_d），发生了域标签切换，即 init 进程创建 ssh 守护进程后，发生了域标签从 initrc_d 到 sshd_d 的切换。

SETE 模型是基于主体的域标签、客体的类型标签、客体的类别和欲实施的操作进行访问控制的，所以，当主体切换了域标签后，或者说，切换了工作域后，它拥有的访问权限随即发生变化，变成与新的域的访问权限相同。5.2.3 节和 5.2.4 节以修改口令的应用为例，比较详细地讨论了进程切换工作域的方法，passwd 进程从 user_d 域切换到 passwd_d 域的主要目的就是要拥有与 passwd_d 域相同的访问权限，以便能够修改 shadow 口令文件。

例 5.18　试举例说明在 SELinux 系统中创建新文件时不切换类型标签和切换类型标签的情形。

解答：文件创建与切换判定如图 5.10 所示，设系统公共临时目录/tmp 的类型为 tmp_t，执行 sort 程序的 sort 进程在 sort_d 域中运行，sort 进程在工作过程中需要在/tmp 目录中创建临时文件/tmp/sorted_result，该文件继承父目录（/tmp）的类型标签（tmp_t），没有发生类型标签切换。

设 syslog 审计进程在 syslogd_d 域中运行，该进程在工作过程中需要在/tmp 目录中创建临时文件/tmp/log.tmp，该文件需要采用 syslog_tmp_t 类型，新文件（/tmp/log.tmp）的类型标签（syslog_tmp_t）不等于父目录（/tmp）的类型标签（tmp_t），发生了类型标签切换。

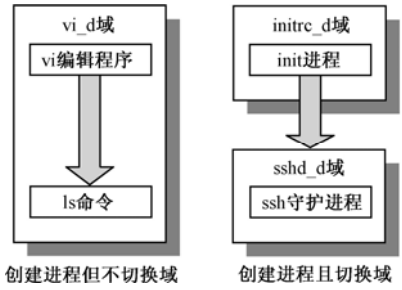


图 5.9　进程创建与域切换

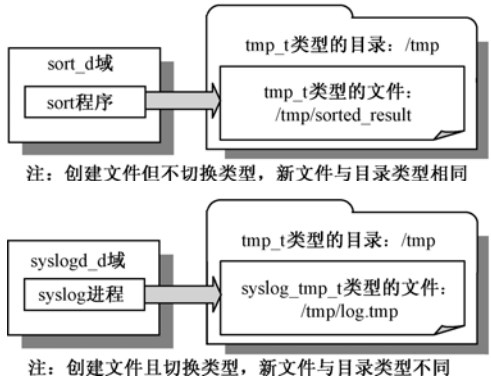


图 5.10　文件创建与切换判定

在例 5.18 中，tmp_t 类型与系统公共临时目录/tmp 相对应，该类型中的文件是对所有域都开放的，所有主体都有权对该类型的文件进行访问，syslog 审计进程创建的/tmp/log.tmp 文件要存放与审计记录相关的信息，只能允许特定域中的主体访问，所以，为该文件选择了特定的 syslog_tmp_t 类型。

例 5.17 说明了创建新进程时域切换的情形，但没有说明域切换的方法，同样，例 5.18 说明了创建新文件时类型切换的情形，但没有说明类型切换的方法。5.2.3 节和 5.2.4 节已经说明了新进程的域切换方法，下面说明新文件的类型切换方法。

与进程的域切换类似，可以使用 type_transition 规则描述文件的类型切换控制，该规则的描述形式如下：

type_transition source_type target_type : file default_type;

其中，source_type 表示进程的域标签，target_type 表示目录的类型标签，default_type 表示文件的类型标签。当在 source_type 域中运行的进程在 target_type 类型的目录中创建新文件时，该规则指示系统把新文件的类型标签切换为 default_type。

例 5.19　试举例说明类型切换规则的定义方法，用于为新创建的文件进行类型切换。

解答：可以定义以下形式的 type_transition 规则：

type_transition syslogd_d tmp_t : file syslog_tmp_t;

该规则要求系统把在 syslogd_d 域中运行的进程在 tmp_t 类型的目录中创建的新文件的类型设置为 syslog_tmp_t。

例 5.18 中的 syslog 审计进程在 syslogd_d 域中运行，它在类型为 tmp_t 的/tmp 目录中创建了新文件/tmp/log.tmp，按照例 5.19 中的规则的指示，系统将尝试把该新文件的类型设置为 syslog_tmp_t。

之所以说系统尝试为新文件设置新的类型标签，是因为该操作是否能够成功取决于访问控制权限是否允许实施该操作。再分析一下例 5.19 中的规则所描述的操作，其中涉

及两个问题：一是文件默认的类型标签应该是 tmp_t，现在要把它改掉，涉及是否允许改的问题；二是要把文件的类型标签设置为 syslog_tmp_t，涉及是否允许取该值的问题。因此，需要从这两个方面考虑来进行相应的授权。

例 5.20　设有描述文件类型切换的 type_transition 规则如下：

type_transition syslogd_d tmp_t : file syslog_tmp_t;

试给出使该规则指示的操作能够成功实施的授权方法。

解答：可以考虑给出以下两个 allow 规则：

allow syslogd_d tmp_t : file { relabelfrom };

allow syslogd_d syslog_tmp_t : file { relabelto };

第一个规则授权在 syslogd_d 域中运行的进程切换在类型为 tmp_t 的目录中创建的文件的类型。第二个规则授权在 syslogd_d 域中运行的进程把文件的类型切换为 syslog_tmp_t。两个规则合在一起，则授权在 syslogd_d 域中运行的进程把在类型为 tmp_t 的目录中创建的文件的类型切换为 syslog_tmp_t。所以，通过这两个 allow 规则的授权，可以使给定的文件类型切换规则指示的操作的成功实施提供必要的访问权限。

从一般意义上考虑，文件的类型切换就是使文件从旧的类型切换为新的类型，在设计文件类型的切换方法时，需要考虑的工作可以归纳为以下几个方面：

- 说明切换意图，指明旧的类型和新的类型；
- 授权改变旧的类型标签；
- 授权赋予新的类型标签；
- 指明实施切换的主体的域标签。

5.3.3　客体类型标签的存储

Linux 操作系统中存在两种性质的客体：临时客体（Transient Object）和永久客体（Persistent Object）。顾名思义，临时客体的生命周期是短暂的，最长不超过操作系统一次从启动（Startup）到关闭（Shutdown）的时间周期，而永久客体的生命周期是无限的，不受操作系统启动和关闭的时间周期的影响。不同性质的客体的类型标签需要用不同的方法进行保存。

进程是最常见的临时客体（也是主体），它们以内核空间中的数据结构的形式存在。SELinux 系统直接把临时客体的类型（或域）标签等安全属性保存在驻留内存的表结构中。

最常见的永久客体是文件和目录。通常，一旦被创建，永久客体就一直存在，直到被删除，它们往往要经历操作系统的多次启动和关闭，所以不能用驻留内存的表结构保存永久客体的类型标签等安全属性，因为操作系统关闭后驻留内存的表结构的内容就不存在了。

通常，永久客体的类型标签等安全属性可以保存在文件系统结构中。Linux 操作系统的 ext2 和 ext3 等标准文件系统提供扩展属性（Extended Attribute）功能，这些功能可以在编译 Linux 操作系统内核时启用。SELinux 系统把永久客体的类型标签等安全属性保存在文件系统的扩展属性结构中。系统运行时，SELinux 系统再把文件系统扩展属性结构中的永久客体的类型标签等安全属性映射到驻留内存的表结构中。

5.3.2 节讨论了给新创建的客体确定类型标签的方法，如何确定已有永久客体的类型标签等安全属性呢？SELinux 系统提供一个 setfiles 程序，支持在安装操作系统时配置文件的类型标签等安全属性，该程序根据一个安全属性配置数据库进行工作，为指定的文件设置指定的类型标签等安全属性，并为其他文件设置默认的类型标签等安全属性。

5.4　SELinux的系统结构设计

本节把目光投向操作系统安全机制的内部结构，了解 SELinux 系统结构设计的整体情况，以便能够从本质上加深对操作系统的增强安全性的认识。SELinux 为 Linux 操作系统的所有内核资源提供增强的访问控制功能，它是按照 Linux 安全模块（LSM，Linux Security Module）框架在 Linux 内核中实现的。首先，有必要简要介绍一下 LSM 框架的基本思想。

5.4.1　Linux安全模块框架

LSM 框架是 Linux 内核支持安全性扩展的方法，其基本思想是通过把安全模块插接到内核中的方式在标准 Linux 访问控制功能的基础上对访问行为进行进一步限制。LSM 在 Linux 内核的系统调用代码中安插了一系列的钩子（hook），这些钩子的安插点位于标准 Linux 的访问权限检查之后，位于访问操作实施之前。图 5.11 所示为 LSM 框架的体系结构。

图 5.11 以用于打开文件的系统调用 open 为例，说明在系统调用中安插钩子的原理。系统调用由用户态进程调用。标准 Linux 中的系统调用 open 的工作过程如下：

（1）通过文件路径名查找描述文件属性的索引节点（Index Node，i 节点）；

（2）检查是否出错；

（3）如果不出错，则根据找到的 i 节点检查自主访问控制（DAC，Discretionary Access Control）权限；

（4）如果 DAC 权限允许访问，则访问文件的 i 节点，即打开文件并返回文件描述符。

对于以上工作过程而言，LSM 框架把一个钩子安插在第（3）步与第（4）步之间，即在 DAC 访问权限检查通过之后且在访问文件的行为开始之前。

钩子的作用是挂接 LSM 的功能模块，从程序设计的角度看，一个钩子实际上就是一个函数调用接口，LSM 功能模块以能够被该接口调用的功能函数的形式出现。

增加了 LSM 机制之后，在系统调用 open 的工作过程中，如果 DAC 权限检查通过，则由与钩子挂接的模块做进一步检查。如果 DAC 权限检查没有通过，则访问已被拒绝，不必再由 LSM 模块进行检查。

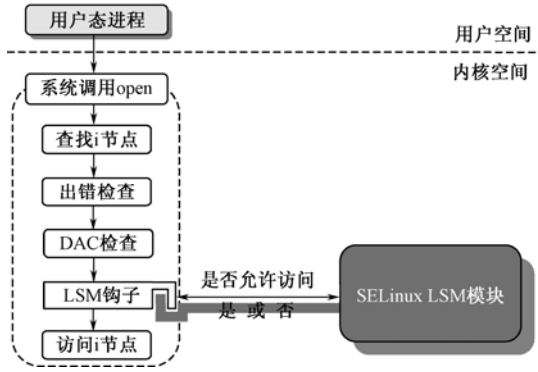


图 5.11　LSM 框架的体系结构

除了 open 系统调用之外，Linux 操作系统中还有其他与安全相关的系统调用，LSM 框架在这些系统调用中都安插了钩子。SELinux 的安全机制由 LSM 模块构成，这些模块通过 LSM 钩子挂接到内核系统调用中。LSM 框架的结构特点决定了 SELinux 的安全机制对标准 Linux 系统的安全机制没有任何负面影响，它只是在标准 Linux 系统安全功能的基础上增加了扩展的安全功能。

5.4.2　SELinux内核体系结构

SELinux 内核体系结构是以 Flask 安全体系结构为基础设计出来的。Flask 体系结构是基于微内核的体系结构，Linux 是非微内核的操作系统，所以，SELinux 是基于微内核的 Flask 体系结构在非微内核系统中的实现。

保持 Flask 体系结构的基本特点，SELinux 内核体系结构由三个主要部分构成，它们是安全服务器、内核客体管理器和访问向量缓存（AVC）等，SELinux 的 LSM 模块的体系结构如图 5.12 所示。

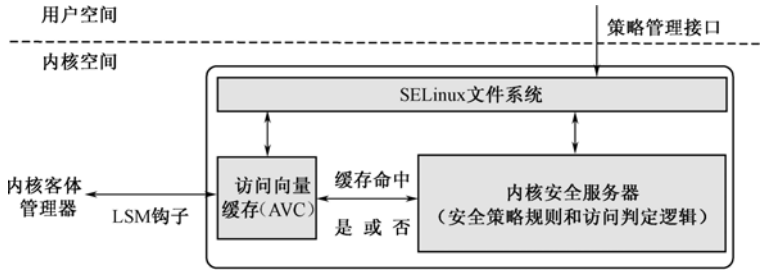


图 5.12　SELinux 的 LSM 模块的体系结构

Flask 体系结构明确区分安全策略判定功能和安全策略实施功能。安全策略判定功

能由安全服务器提供，安全服务器这个说法隐含着微内核结构的浓厚色彩，在微内核结构中，安全策略的判定由用户空间中的服务器实施。在 SELinux 系统中，面向内核客体的安全服务器由 SELinux 的 LSM 模块构成。安全服务器使用的安全策略由一系列安全规则表示，这些规则通过策略管理接口装入到内核中。不同的系统可以配备不同的安全规则，因此，SELinux 对不同的系统安全目标有很强的适应性。

安全判定与安全实施分离的体系结构设计对安全策略具有灵活的支持能力，因为可以用实现全新的安全策略的服务器替代原有服务器，而不需要修改体系结构中的其他部分。不过，在 SELinux 系统中，没必要更换新的安全服务器，因为 SELinux 的安全服务器实现的 SETE 模型的灵活性几乎足以支持任何类型的访问控制策略。

客体管理器在它所管理的资源上实施安全服务器所提供的安全策略判定结果。在 Linux 内核中，负责创建和管理内核客体的内核子系统就是客体管理器，如文件系统、进程管理系统和 System V 的进程间通信（IPC，Interprocess Communication）系统等都是内核客体管理器的实例。

在 LSM 体系结构中，客体管理器由 LSM 钩子表示，这些钩子分布在内核的多个子系统中，它们调用 SELinux 的 LSM 模块进行访问判定，通过允许或拒绝对内核资源的访问来实施这些访问判定的结果。

SELinux 体系结构的第三个主要部分是 AVC。AVC 缓存安全服务器生成的访问判定结果供以后在进行访问权限检查时使用，这样能显著提高访问许可验证的性能。同时，AVC 为 LSM 钩子提供了与 SELinux 的接口，进而，也成为内核客体管理器与安全服务器之间的接口。

至此，可以结合图 5.12 把 SELinux 内核体系结构的三个主要部分联系起来，说明该体系结构的工作过程。当客体管理器通过 LSM 钩子调用 SELinux 的 LSM 模块进行访问控制判定时，LSM 模块首先在 AVC 中查找与访问请求相符的 AV，如果命中，则把它传给客体管理器，如果不能命中，则交由安全服务器根据访问控制策略进行判定，安全服务器完成判定后，生成 AV 结果，把它交给 AVC，进而传给客体管理器。客体管理器根据得到的结果确定是否允许实施访问操作。安全策略在用户空间中定义，然后通过策略管理接口装入到内核中，供安全服务器使用。

每当装入安全策略时，AVC 都被置为无效，以求维持缓存的访问控制判定内容与安全策略之间的一致性。需要一提的是，SELinux 并没有支持在安全策略变化时完全撤销访问控制权限，不过，它在这方面并不比标准 Linux 差，因为标准 Linux 根本不提供撤销访问控制权限的功能。

在标准 Linux 中，只是在打开文件时检查进程对文件的访问权限。当一个进程通过系统调用 open 打开一个文件获得了其文件描述符后，该进程就可以一直通过该文件描述符访问该文件，不管其后该文件的访问权限属性是否发生变化。

例 5.21　设 P 是 Linux 系统中的一个进程，P 的有效身份为 alice，alice 不属于 root 组，在 T_1 时刻，文件 froot 的权限信息如下：

rw-r--r--　root　root　..... froot

在 T_2 时刻，文件 froot 的权限信息被修改为如下形式：

rw-r-----　root　root　..... froot

设 $T_1< T_2< T_3$，在 T_1 时刻，进程 P 欲以“读”方式对文件 froot 执行 open 系统调用，在 T_3 时刻，进程 P 欲对文件 froot 执行 read 系统调用。请问 open 和 read 系统调用是否能成功执行？

解答：在 T_1 时刻，用户 alice 对文件 froot 有“读”权限，进程 P 的有效身份为 alice，所以，进程 P 对文件 froot 有“读”权限，open 系统调用能成功执行，设它返回的文件描述符为 fdr。在 T_3 时刻，用户 alice 没对文件 froot 有“读”权限，因而，进程 P 对文件 froot 没有“读”权限，但是，进程 P 是通过 fdr 对文件 froot 执行 read 系统调用的，而且，不需要进行访问权限检查，所以，read 系统调用能成功地执行。

例 5.21 显示，在 T_3 时刻，虽然进程 P 对文件 froot 没有“读”权限，但它能成功地执行对该文件的“读”操作。这反映出在标准 Linux 系统中访问操作与访问授权之间存在一定的不一致性。

在 SELinux 中，不仅在打开文件时检查访问权限，在所有的访问尝试中都要检查访问权限。例如，对已打开的文件实施读操作前也要检查访问权限。如果在打开文件时，文件是可读的，那么，打开操作是成功的，但是，如果在读操作前，文件已不可读，那么，在 SELinux 中，读操作是不能执行的。所以，SELinux 能较好地提供撤销访问权限的支持。显然，与标准 Linux 不同，在 SELinux 中，拥有文件描述符并不意味着可以对文件进行访问。

例 5.22　设 P 是 SELinux 系统中的一个进程，P 的有效身份为 alice，alice 不属于 root 组，P 在 p_d 域中运行，在 T_1 时刻，p_d 域对类型为 fr_t 的文件有“读”权限，fr_t 类型的文件 froot 的权限信息如下：

rw-r--r--　root　root　..... froot

在 T_2 时刻，p_d 域对类型为 fr_t 的文件的“读”权限被撤销。

设 $T_1<T_2<T_3$，在 T_1 时刻，进程 P 欲以“读”方式对文件 froot 执行 open 系统调用；在 T_3 时刻，进程 P 欲对文件 froot 执行 read 系统调用。请问 open 和 read 系统调用是否能成功执行？

解答：在 T_1 时刻，P 在标准 Linux 的 DAC 模型中和在 SELinux 的 SETE 模型中对文件 froot 都有“读”权限，所以，open 系统调用能成功执行。在 T_3 时刻，由于 p_d 域对类型为 fr_t 的文件的“读”权限已被撤销，所以，P 在 SELinux 的 SETE 模型中对文件 froot 没有“读”权限，因为 read 系统调用中安插有 LSM 钩子，它要调用 SELinux 的 LSM 模

块进行访问权限检查，检查结果必然是拒绝访问，故此，read 系统调用不能成功地执行。

例 5.22 显示，SELinux 能较好地提供撤销访问权限的支持，进而，能较好地实现访问操作与访问授权之间的一致性。

当然，SELinux 也并非为所有客体都提供了撤销访问权限的支持。对于内存映射文件和面向连接的套接字等客体，也只有在首次访问前才进行访问权限检查，如果检查通过，以后的访问并没有再进行检查，所以，对它们的访问权限的撤销还没有得到支持。

5.4.3　用户空间的客体管理器

SELinux 体系结构的突出特点之一是既支持对内核资源实施访问控制也支持对用户空间的资源实施访问控制。这是自然的，因为该体系结构源自 Flask 体系结构，而 Flask 体系结构是面向微内核的，在微内核系统中，大部分的资源管理是由用户空间的服务器实施的。

在 Linux 系统中，X 窗口服务器和数据库服务器等都是用户空间服务器的实例，这些服务器对其资源的访问控制功能可以进行进一步增强，它们都提供有助于实施强制安全性的抽象资源，如窗口和表等。

5.4.2 节介绍了 SELinux 对内核资源实施访问控制的方法，下面讨论在 SELinux 体系结构中对用户空间的资源实施访问控制的支持方法。

例 5.23　试以 SELinux 的内核体系结构为基础，给出一个设计方案，用于支持对用户空间的资源实施访问控制。

解答：仿照内核中的客体管理器，在用户空间设立相应的客体管理器，用于实施与用户空间的访问请求对应的访问判定结果。用户空间的客体管理器通过内核的安全服务器进行访问控制判定，基于内核安全服务器的用户空间客体管理器如图 5.13 所示。用户空间的客体管理器无法使用内核的 AVC，可以在用户空间的客体管理器中设立 AVC，供该客体管理器使用。在对用户空间的资源进行访问前，用户空间的客体管理器在它自己的 AVC 中查找访问判定结果，如果能命中，则根据它确定是否允许实施访问，如果不能命中，则向内核安全服务器发判定请求，等判定结果传到用户空间的客体管理器的 AVC 中再确定访问许可。内核中的 AVC 是内核安全服务器与 LSM 钩子间的接口，在用户空间不存在 LSM 钩子，可以通过函数库 libselinux 实现用户空间的 AVC 的接口支持。用户空间的 AVC 处理设有命中结果的查询，并为用户空间的客体管理器向内核安全服务器发判定请求。

例 5.23 提供的方案比较简单和直接，但存在一定的不足。首先，SELinux 的 LSM 模块只为内核资源定义了客体类别，SETE 模型根据客体类别进行访问控制，因此，用户空间的每个客体管理器必须为它管理的资源定义客体类别，如数据库服务器可能需要定义数据库、表、模式、记录等客体类别。

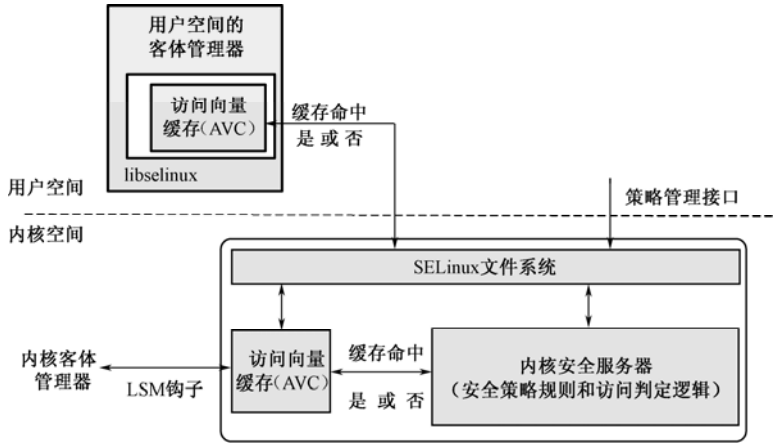


图 5.13　基于内核安全服务器的用户空间客体管理器

用户空间的客体管理器定义的客体类别不能与内核的客体类别冲突。例如，不能采用相同的标识等。用户空间的两个不同的客体管理器定义的客体类别也不能冲突，因为所有的客体类别都要供同一个内核安全服务器使用。

另外，内核安全服务器需要管理为用户空间的客体管理器设置的针对用户空间客体类别的访问控制策略，迫使系统要在内核中保存不属于内核的信息，导致内核的存储开销增加，并给内核访问控制判定的开销带来负面影响。

例 5.24　试以例 5.23 的方案为基础，给出一个改进的设计方案，用于支持对用户空间的资源实施访问控制。

解答：仿照内核中的安全服务器，在用户空间设立相应的安全服务器，用于为用户空间的客体管理器提供访问控制判定。增设用户空间的安全服务器后，可以把系统中的安全策略划分成两个部分：一部分是内核安全服务器要处理的安全策略，简称内核策略；另一部分是用户空间的安全服务器要处理的安全策略，简称用户策略。由于安全策略是在用户空间中定义的，可以在用户空间中设立一个策略管理服务器，用于对 SELinux 系统中的所有安全策略进行总体处理，从中区分出内核策略和用户策略，把内核策略装入到内核安全服务器中，把用户策略装入到用户空间的安全服务器中。SELinux 策略服务器体系结构如图 5.14 所示，它的主要特点是设立了策略管理器和用户空间的安全服务器。

安全策略是安全服务器进行访问控制判定的依据，是安全服务器要访问的特殊资源，管理这些特殊资源的任务由策略管理服务器承担，从这个意义上说，策略管理服务器属于用户空间的客体管理器，它负责为策略资源创建客体类别，实施对策略资源的细粒度的访问控制策略。

对策略资源的粗粒度访问控制只能支持基于整个策略文件的访问权限。例如，要么授权对整个策略文件进行“写”操作，要么禁止对整个策略文件进行“写”操作，无法授权对策略文件的部分内容进行“写”操作。

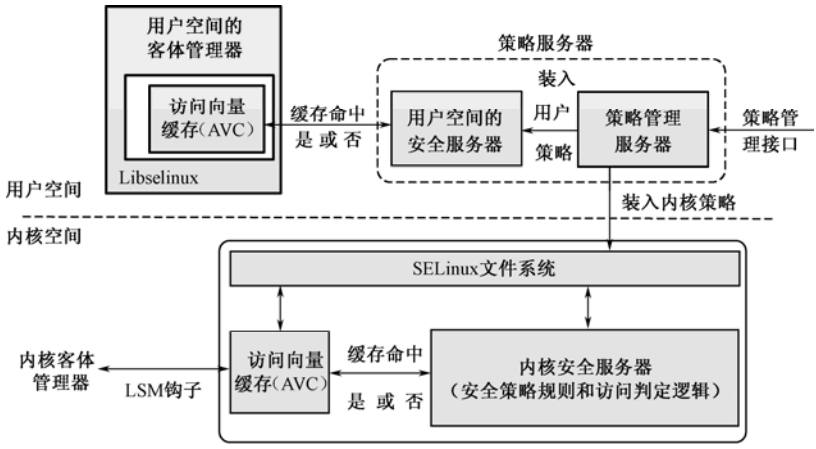


图 5.14　SELinux 策略服务器体系结构

对策略资源的细粒度访问控制支持基于特定策略资源的访问权限。例如，可以授权数据库服务器修改针对其所管理的客体类别和类型的 SETE 规则，而禁止其修改针对内核客体类别和类型的 SETE 规则。

例 5.24 给出的策略管理服务器可用于在 SELinux 中实现可装载策略模块（Loadable Policy Modules）功能，为安全策略的配置和管理提供灵活的支持。

例 5.24 中的方案把内核策略和用户策略区分开后，内核安全服务器只需存储和处理内核策略中的规则和客体类别，减小了内核安全服务器的存储开销，优化了内核安全服务器访问判定的性能。同样，用户策略的规则和客体类别存储在用户空间的安全服务器中，由用户空间的安全服务器进行用户空间的资源访问判定，能提高用户空间的客体管理器的相应效率。

总的来说，相对于例 5.23 中的方案而言，例 5.24 中的方案一方面能提高安全策略管理的灵活性，另一方面能优化内核空间和用户空间的访问判定的性能。另外，策略管理服务器运行在用户空间中，这有利于进一步扩充以接受来自网络的访问，从而实现分布式的安全策略管理。

5.5　SELinux的策略语言

TE 模型只提供了访问控制框架，并未提供访问控制策略，要应用该模型进行访问控制，必须为每个系统设计相应的访问控制内容。DTE 模型对 TE 模型的重要改进之一是设计了 DTEL，用于描述访问控制策略。SELinux 访问控制模型的核心是 DTE 模型（即 SETE 模型），了解 SELinux 的策略语言（SEPL）是掌握 SELinux 的强制访问控制方法的关键。

前面曾经介绍过运用 SEPL 的局部功能在 SELinux 系统中进行访问控制的基本方

法，本节主要从宏观上介绍 SEPL 的整体架构。

5.5.1　SEPL策略源文件及其编译

在 SELinux 系统中，用 SEPL 描述的安全策略定义和配置存放在一个文件中，该文件称为策略源文件，策略源文件的典型文件名是 policy.conf。SEPL 策略源文件的组织结构如图 5.15 所示。



图 5.15　SEPL 策略源文件的组织结构

策略源文件的内容由客体类别与权限、SETE 模型等的描述、约束和客体标签描述等几部分组成。

策略源文件的第一部分是客体类别与权限部分，这个部分定义客体的类别，并定义每个客体类别的访问权限。内核客体的类别和权限在 SELinux 的内核源代码中有确定的定义，编写安全策略时，通常不要修改这些客体类别和权限的定义。

例 5.25　下面的第一个语句声明一个名为 dir 的客体类别，第二个语句把已声明的 dir 类别与名为 search、add_name 和 remove_name 的三个权限关联起来：

```
class dir
class dir { search add_name remove_name }
```

例 5.26　下面的语句声明一个名为 ping_t 的类型，并把它与名为 domain、privlog 和 nscd_client_domain 的三个属性关联起来：

```
type ping_t, domain, privlog, nscd_client_domain;
```

例 5.27　下面的语句声明一个名为 sysadm_r 的角色，并把它与名为 ping_t 的域类型关联起来：

```
role sysadm_r types ping_t;
```

策略源文件的第二部分是 SETE 模型等的描述部分，这个部分定义访问控制策略，核心是 SETE 模型的访问控制策略。这是策略编写人员要花费大部分的策略编写时间去编写的部分，它包含所有的类型声明和所有的 SETE 规则（包括所有的 allow 规则、type_transition 规则等），通常包含几千个类型声明和几万个 SETE 规则。除了 SETE 模型外，SELinux 提供对基于角色的访问控制（RBAC，Role Based Access Control）等模型的支持，策略源文件的这个部分也包含关于角色和用户的规则和声明。

例 5.28　下面的 allow 规则给 ping_t 域的主体授予对 self 类型的 unix_stream_socket 类别的客体的 create_socket_perms 权限，即授权 ping_t 域的进程创建 self 类型的 UNIX 套接字：

```
allow ping_t self:unix_stream_socket create_socket_perms
```

策略源文件的第三部分是约束部分，这个部分在 SETE 规则的控制范围之外对 SETE 策略加以进一步限制。SELinux 也提供对基于 BLP 模型的多级安全策略（MLS，Multilevel Security）模型的支持，MLS 策略是通过一组约束来实现的。

例 5.29　以下是一个约束，当一个进程请求对一个文件进行 create、relabelto 或 relabelfrom 操作时，它要求文件的用户属性必须匹配进程的用户属性，或者进程的工作域必须拥有 privowner 属性：

```
constrain file { create relabelto relabelfrom }
( u1 == u2 or t1 == privowner );
```

策略源文件的最后一个部分是客体标签描述部分。为了实施 SELinux 的访问控制，必须给所有的客体都分配类型标签，这个部分描述文件系统中的客体的标签分配方法，也描述运行期间创建的临时客体的标签分配规则。

例 5.30　下面的第一个规则描述文件类型标签的分配方法，第二个规则描述进程域标签的分配方法：

```
type_transition syslogd_d tmp_t : file syslog_tmp_t;
type_transition user_d passwd_exec_t : process passwd_d;
```

根据第一个规则，在 syslogd_d 域中运行的进程在 tmp_t 类型的目录中创建的文件的类型标签将被设为 syslog_tmp_t。根据第二个规则，当 user_d 域中的进程通过 exec 系统调用执行 passwd_exec_t 类型的入口点程序时，该进程的域标签将被设为 passwd_d。

策略源文件是便于人理解的一种策略描述格式，它不适合于计算机理解，计算机的特长是理解二进制信息，所以，需要把策略源文件编译成策略二进制文件，供安全服务器使用。

SELinux 系统的 checkpolicy 程序用于实施对策略源文件的编译工作，它检查策略源文件的语法和语义的正确性，如果语法和语义都正确，则生成策略二进制文件。

SELinux 系统的内核策略装载程序 load_policy 能够识别策略二进制文件，它负责把二进制格式的安全策略装入到内核安全服务器中。

5.5.2　安全策略的构造与装载

在讨论安全策略的构造与装载方法之前，首先了解一下 SELinux 系统中使用的安全策略的基本形式。

SELinux 的常用策略形式是单体策略（Monolithic Policy）。即，在系统内部，只使用一个二进制格式的策略文件，该文件描述所有的安全策略。可是，SELinux 的策略总是非常庞大和复杂的，就像一个大型软件系统，一下子构造一个庞大的策略文件很不容易，可行的途径是采取模块化的方法，首先在源文件一级编写很多小的策略模块，然后

再把这些策略模块组合成完整的策略文件。

用 SEPL 编写的策略模块称为策略源模块，可以综合使用 shell 脚本、m4 宏和 make 工具，把策略源模块组合成单一的策略源文件（即 policy.conf 文件），该文件可由 checkpolicy 程序编译成内核能够识别策略二进制文件。在单体策略方法中，在编写策略的过程中，可以编写很多策略模块，但二进制的策略文件是单一的，没有二进制的策略模块。

与单体策略不同的一种形式是可装载策略模块（Loadable Policy Modules）的形式。在该形式中，不采用单一的策略二进制文件描述所有的安全策略，而采用多个二进制的策略模块来描述安全策略。在众多的策略模块中，有一个是基础模块，该模块描述安全策略的核心内容，只包含操作系统的核心规则。基础策略模块采用与单体策略类似的方法创建和装入内核，其他的策略模块都设计为可装载策略模块，由模块编译程序 checkmodule 独立编译，需要时再装入安全服务器。可装载策略模块方法是实现如图 5.14 所示的策略服务器的基础。

下面以单体策略和内核安全服务器为例，讨论 SELinux 系统中安全策略的构造与装载的方法，该方法可以用图 5.16 表示。

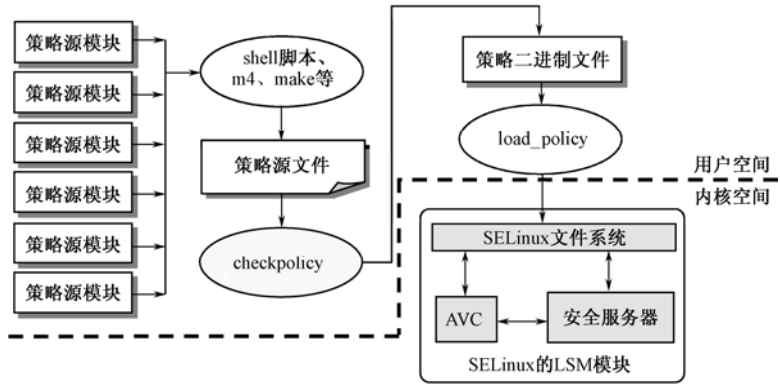


图 5.16　SELinux 安全策略的构造与装载方法

用 SEPL 构造 SELinux 系统中的安全策略并把安全策略装载到内核安全服务器的方法如下：

（1）用 SEPL 编写策略源模块（源模块在源模块文件中描述，描述中可以使用 shell 脚本和 m4 宏）；

（2）借助 shell 脚本、m4 宏和 make 工具把所有策略源模块（表现形式是源模块文件）组合成单一策略源文件（policy.conf）；

（3）用 checkpolicy 程序把单一策略源文件编译成单一策略二进制文件；

（4）用 load_policy 程序把单一策略二进制文件装入到内核安全服务器中。

内核中的安全服务器根据已装入的策略二进制文件识别出系统实施的访问控制策

略，并以它们为依据进行访问控制判定。

5.5.3　策略源模块样例

本节从宏观上介绍 SEPL 的整体架构，不是介绍 SEPL 的细节。为了初步建立对用 SEPL 编写的策略源文件的感性认识，这里给出一个策略源模块的例子，读者可以从中了解编写策略源模块的基本思想，暂时无须弄清其中的所有细节。

例 5.31　以下是 SELinux 系统提供的“严格策略样例”中为 ping 程序编写的策略模块的源文件的内容（行号是为了说明方便而加的，源文件名为 ping.te）：

```
1   type ping_t, domain, privlog, nscd_client_domain;
2   role sysadm_r types ping_t;
3   role system_r types ping_t;
4   in_user_role(ping_t)
5   type ping_exec_t, file_type, sysadmfile, exec_type;
6
7   # 运行本程序时切换到本域中.
8   domain_auto_trans(sysadm_t, ping_exec_t, ping_t)
9   domain_auto_trans(initrc_t, ping_exec_t, ping_t)
10  bool user_ping false;
11  if (user_ping) {
12      domain_auto_trans(unpriv_userdomain, ping_exec_t, ping_t)
13      # 允许访问终端.
14      allow ping_t { ttyfile ptyfile }:chr_file rw_file_perms;
15      ifdef('gnome-pty-helper.te', 'allow ping_t gphdomain:fd use;')
16  }
17
18  uses_shlib(ping_t)
19  can_network_client(ping_t)
20  can_resolve(ping_t)
21  allow ping_t dns_port_t:tcp_socket name_connect;
22  can_ypbind(ping_t)
23  allow ping_t etc_t:file { getattr read };
24  allow ping_t self:unix_stream_socket create_socket_perms;
25
26  # 使 ping 创建原始 ICMP 包.
27  allow ping_t self:rawip_socket {create ioctl read write bind getopt setopt };
28
```

```
29  # 使用权能.
30  allow ping_t self:capability { net_raw setuid };
31
32  # 访问终端.
33  allow ping_t admin_tty_type:chr_file rw_file_perms;
34  allow ping_t privfd:fd use;
35  dontaudit ping_t fs_t:filesystem getattr;
36
37  # 尝试访问/var/run.
38  dontaudit ping_t var_t:dir search;
39  ifdef(`hide_broken_symptoms', `
40    dontaudit ping_t init_t:fd use;
41  ')
```

第 4、8、9、12、18～20 和 22 行中给出的是 m4 宏语句，不是 SEPL 语句，宏语句在编译时展开为 SEPL 语句。

第 1 行和第 5 行分别定义 ping_t 和 ping_exec_t 两个类型，其中，ping_t 是为 ping 进程定义的域类型，ping_exec_t 是为 ping 可执行程序文件定义的类型。

第 2 行和第 3 行分别把角色 sysadm_r 和 system_r 与域类型 ping_t 关联起来，第 4 行通过宏描述角色与域类型 ping_t 的关联。

第 8 行和第 9 行通过宏描述进程的域切换，实际上，它们分别说明当域 sysadm_t 和 initrc_t 中的进程通过 exec 系统调用执行 ping_exec_t 类型的入口点程序时，该进程将切换到域 ping_t 中，即那两个域中的进程执行 ping 程序时切换到域 ping_t 中。

第 10 行定义布尔变量 user_ping，第 11～16 行利用该布尔变量定义了一个条件策略语句段。

第 18 行给域 ping_t 授予访问共享库的权限，第 19～24 行和第 27 行给该域授予访问网络和系统资源的权限。

第 30 行中的关键词 self 表示源类型（域）自己，即等同于域 ping_t，该规则给 ping_t 类型（域）授予对 ping_t 类型的 capability 类别进行访问的权限，capability 类别控制 Linux 的权能。

第 33 和第 34 行的规则给域 ping_t 授予利用终端设备显示输出信息的权限。剩余的其他语句描述对审计事件的响应方法，说明指定操作遭拒绝时无须审计（可参见 5.3.1 节）。

例 5.31 中出现了有关角色的描述，结合这一点，我们最后简单提一下 SELinux 系统中的安全环境（Security Context）的概念。

本章的重点是 SETE 模型，而不是对 SELinux 系统进行全面的介绍，所以，前面的

讨论主要关注主体和客体的类型属性（对于主体而言，也就是域属性）。实际上，SELinux 用安全环境来表示主体和客体的安全属性。每个主体和客体都有一个安全环境，每个安全环境包含用户、角色和类型三个元素，安全环境的一般表示形式为

```
user:role:type
```

其中，user 代表用户标识，role 代表角色标识，type 代表类型标识。例如，以下可以是一个安全环境的值：

```
alice:system_r:ping_t
```

其中的用户标识为 alice，角色标识为 system_r，类型标识为 ping_t。注意：安全环境中的用户标识与标准 Linux 系统账户中的用户标识不是一回事，它们是两套相互独立的体系，前者由 SELinux 使用，后者由标准 Linux 使用，它们之间没有必然的联系。

进程安全环境中的用户和角色元素分别表示进程的用户身份和角色身份。客体安全环境中的角色元素没有意义，习惯上被设置为 object_r，用户元素通常取创建该客体的进程的用户标识。

SETE 模型在访问控制中主要使用安全环境中的类型元素，只有在约束中，才用到用户元素（见例 5.29）。

SELinux 系统把安全环境存放在一张表中，并用安全标识符（SID，Security Identifier）来标识每一个表项。每个 SID 是一个整数，SELinux 基于 SID 进行访问控制判定。

5.6　本章小结

本章的目标是通过一个典型的模型（DTE 模型）和一个典型的系统（SELinux 系统）介绍操作系统实现强制访问控制的一种方法，进而，以强制访问控制为实例，介绍操作系统的增强安全性。基于这个目标，依次介绍了 TE 模型、DTE 模型和 SELinux 系统实现的对应模型（本章称其为 SETE 模型），以此为主线，对增强操作系统安全性的基本技术和方法进行了探讨，主要内容包括 TE 模型与 DTE 模型、SELinux 实现的 TE 模型、访问判定与切换判定、SELinux 的系统结构设计和 SELinux 的策略语言等。

TE 模型是一个可以实现强制访问控制的模型，是一个面向二维表的访问控制模型，它把主体划分为域，把客体划分为类型，用 DDT 表定义域与类型间的授权关系，以表中的行描述域，以表中的列描述类型，以行与列的交叉点描述访问权限。TE 模型可以实现应用系统的有效隔离。

DTE 模型是改进版的 TE 模型，DTE 模型对 TE 模型的改进主要体现在两个方面：一是使用高级语言形式的 DTEL 描述访问控制策略，取代用二维表描述访问控制策略的方法；二是充分利用文件系统的层次结构和进程的层次结构表示客体和主体的安全属性，使安全属性的定义能够反映系统的内在结构。

DTEL 提供类型描述、类型赋值、域描述和初始域设定等方面的功能，类型描述功

能定义客体类型，类型赋值功能把客体与客体类型关联起来，域描述功能定义主体域并设定域对类型的访问权限及域对域的访问权限，初始域设定功能设定第一个进程的工作域，进而隐含地确定所有进程的工作域。

SETE 模型对 DTE 模型进行了扩充和发展，与 DTE 模型相比，SETE 模型具有类型的细分和权限的细化等方面的突出特点。SETE 模型在类型概念的基础上，增加了客体类别的概念，针对类型和类别确定访问权限。SETE 模型为客体定义了几十个类别，为每个类别定义了相应的访问权限。

在 SELinux 中，只有明确授权的访问才是允许的访问。SETE 模型通过 SEPL 描述访问控制策略，确定访问控制的授权方法，allow 规则是描述访问控制授权的基本方法。allow 规则包含源类型、目标类型、客体类别和访问权限四个元素，它定义源类型对目标类型的客体类别拥有的访问权限。

一个进程运行时所在的域称为该进程的工作域，一个进程的工作域在不同的时刻可以不同。切换进程的工作域的主要目的是为了使进程获得新的访问权限。进程可以在执行 exec 系统调用装载新的域的入口点程序时完成工作域的切换。只有在明确授权的情况下，进程才能切换工作域，切换授权通过 allow 规则定义。在满足工作域切换条件的情况下，可以通过 type_transition 规则实施进程工作域的自动切换。

SELinux 实现的判定有访问判定和切换判定两种基本形式。访问判定确定给定的主体能否在给定的客体上实施给定的操作，切换判定确定是否需要切换新创建的进程和文件的类型标签，对于进程而言，类型指的就是域。SELinux 系统用 AV 表示访问判定的结果，用 AVC 实现 AV 信息的缓存。

一般情况下，新进程的域标签等于父进程的域标签，新文件的类型标签等于父目录的类型标签。有时需要给新的主体或新的客体分配新的标签，切换判定就是要确定是否需要给新的主体或新的客体分配新的标签及新的标签应该取什么值。与进程的工作域切换类似，用 allow 规则定义文件的类型切换授权，用 type_transition 规则定义文件的类型切换操作。SELinux 系统把永久客体的类型标签等安全属性保存在文件系统的扩展属性结构中。

SELinux 内核体系结构是以 Flask 安全体系结构为基础设计出来的，Flask 安全体系结构是基于微内核的体系结构，SELinux 是基于微内核的 Flask 安全体系结构在非微内核系统中的实现。SELinux 是按照 LSM 框架在 Linux 内核中实现的，LSM 框架是 Linux 内核支持安全性扩展的方法，其基本思想是通过钩子把安全模块插接到内核中，在标准 Linux 访问控制功能的基础上，对访问行为实施进一步限制。

SELinux 内核体系结构由安全服务器、客体管理器和 AVC 三个主要部分构成，它明确区分安全策略判定功能和安全策略实施功能，安全策略的判定由安全服务器执行，安全策略的实施由客体管理器执行。AVC 缓存安全服务器生成的访问判定结果，能显著提高访问许可验证的性能。SELinux 提供一定的访问权限撤销功能。

SELinux 的 LSM 钩子机制主要实现对内核资源的控制。通过用户空间的客体管理器，SELinux 可以对用户空间的资源实施访问控制。用户空间的客体管理器可以直接利用原有的内核安全服务器进行访问控制判定，也可以利用新增的用户空间的安全服务器进行访问控制判定。用户空间的策略管理服务器有利于实现分布式的安全策略管理。

SELinux 的 SEPL 描述的安全策略定义和配置构成策略源文件 policy.conf，它的内容由客体类别与权限、SETE 模型等的描述、约束和客体标签描述等几个部分组成，它定义了 SELinux 中的所有安全策略。checkpolicy 程序把策略源文件编译成策略二进制文件，load_policy 程序把策略二进制文件装入到内核安全服务器中。

SELinux 的策略非常庞大和复杂，安全策略的编写通常采取模块化的方法，首先用 SEPL 编写策略源模块文件，再借助 shell 脚本、m4 宏和 make 工具等把策略源模块文件组合成策略源文件。

SELinux 用安全环境来表示主体和客体的安全属性。每个主体和客体都有一个安全环境，每个安全环境包含用户、角色和类型三个元素。SELinux 系统把安全环境存放在一张表中，并用 SID 来标识每个表项。每个 SID 是一个整数，SELinux 基于 SID 进行访问控制判定。



习题 5

1. 简要说明 TE 模型的基本思想，并分析该思想与访问控制矩阵思想的相似之处和不同之处。

2. DTE 模型与 TE 模型的区别主要体现在哪些方面？

3. 举例说明 DTE 模型是如何利用文件系统的层次结构和进程的层次结构来表示客体和主体的安全属性的。

4. 以进程对文件的访问为例，简要说明 DTE 模型的 DTEL 主要是从哪几个方面描述访问控制规则的。

5. 在 DTE 模型中，什么是域的入口点？什么是域切换？

6. SETE 模型与 DTE 模型的区别主要体现在哪些方面？

7. 简要说明 SETE 模型的 SEPL 描述访问控制规则的基本方法，请举例加以说明。

8. 在 SETE 模型中，“进程切换工作域”是什么意思？“进程切换工作域”的主要目的是什么？

9. 在 SETE 模型中，进程为实现工作域切换必须满足哪些条件？简要说明授权进程进行工作域切换的基本方法。

10. 在 SETE 模型中，提供类型切换措施的目的是什么？

11. 在 SELinux 系统中，什么是访问判定？什么是访问向量（AV）？简要说明访问判定的基本方法。

12. 在 SELinux 系统中，什么是切换判定？简要说明切换判定的基本方法。

13. 在 SELinux 系统中，什么是临时客体？什么是永久客体？试举例加以说明。临时客体和永久客体的 SETE 模型相关标签是如何存储的？

14. Linux 系统提供 LSM 框架的目的是什么？简要说明 LSM 的基本思想。

15. SELinux 系统的内核体系结构主要由哪几部分构成？各部分主要分别负责承担什么任务？

16. 在 SELinux 系统的体系结构设计中，为什么需要用户空间的客体服务器？引入用户空间的安全服务器有什么好处？引入用户空间的策略管理服务器有什么好处？

17. 在 SELinux 系统中，什么是策略源模块？什么是策略源文件？它们与 SELinux 实施的访问控制策略有什么关系？

第 6 章
数据库系统的基础安全性

本章首先介绍数据库系统安全性的整体概况，然后以关系数据库系统为代表，着重从关系数据库自主访问授权、基于视图的访问控制、基于角色的访问控制和数据库数据的推理控制等方面介绍数据库系统安全性的基本思想、技术和方法。

6.1　数据库系统安全概貌

从宏观上了解数据库系统安全性的整体情况，对于学习具体的数据库安全基础知识非常有意义，这有助于我们弄清所要学习的知识点在解决数据库系统安全问题中的地位和作用。数据库系统安全是信息系统安全的重要组成部分，一方面，具有信息安全的共同特性，另一方面，具有自身鲜明的特点。首先，可以从信息安全基本要素的角度去认识数据库系统安全中的问题，同时，可以把注意力集中到数据库系统安全的特定领域，去了解其中的典型研究课题，从而形成对数据库系统安全的整体认识。

6.1.1　数据库安全的经典要素观

通过前面的学习，我们已经知道，CIA 是信息安全中经典的三个基本要素，即 C—机密性（Confidentiality），I—完整性（Integrity），A—可用性（Availability）。结合学科发展的特点，我们再考虑另一个要素，即 P—私密性（Privacy）。在本节中，我们从 CIAP 四要素的角度去了解数据库系统安全的知识体系。私密性问题也称为隐私问题，实现私密性就是实现隐私保护，本章把私密性和隐私作为等价概念使用。

1．数据库数据的CIA

根据信息安全的经典要素，数据库系统受到的安全威胁可以划分为数据的非授权查询、数据的不恰当修改和数据的不可用。数据的非授权查询会导致信息泄露给无权访问这些信息的用户。数据的不恰当修改，不管是有意的还是无意的，会导致数据库陷入非正常状态。数据的不可用会导致应用系统的关键功能无法正常实现。

所以，数据库系统安全的完整解决方案必须符合机密性、完整性和可用性的要求。机密性是指保护数据免遭非授权的泄露。完整性是指防止数据受到非授权的或非适当的修改。可用性是指确保数据库系统正常提供数据服务，这既包括防止软硬件出错和实现软硬件出错时的系统恢复，也包括防止出现恶意的数据访问拒绝。

大量的应用环境都有机密性、完整性和可用性的需求。

例 6.1　在一个工资信息数据库系统中，实现以下需求是很重要的：

（1）防止将员工的个人工资透露给非授权的用户；

（2）确保只有获得合理授权的用户才能修改工资数据；

（3）确保在规定的发放工资时间及时打印工资单信息。

例 6.2　在一个航空公司的网站系统中，实现以下需求是很重要的：

（1）保证客户的订票信息只提供给订票者本人；

（2）防止客户的订票信息被随意修改；

（3）保障系统的航班信息服务和订票服务时刻正常工作。

2．数据库数据的私密性

随着社会的进步和应用的发展，除了机密性、完整性和可用性外，在数据库系统安全领域，私密性需求也日显突出。

虽然私密性与机密性具有非常相近的含义，但是，它们是两个具有明显区别的需求。实现信息机密性的技术可以用于实现私密性，同时，保障私密性还需要其他技术。例如，需要一些用于采集和记录用户的允诺信息的机制。另外，可以把数据封存起来，禁止任何访问，从而获取机密性，然而，私密性则要求，即便数据已经暴露，也不能够被他人利用。换句话说，私密性要求数据只能用于用户认可的目的，不能用做他途。

3．数据库数据安全性支持机制

数据库数据的安全性由数据库管理系统（DBMS，Database Management System）的多种组件来实现支持。

数据的机密性主要由访问控制机制实现。每当一个主体要对一个数据客体进行访问时，访问控制机制就根据一组授权来检查用户的访问权限。一个授权描述一个主体是否能够对一个客体进行一个特定的操作，它通常是由安全管理员根据应用单位的访问控制策略定义的。

还可以使用加密技术来进一步增强数据的机密性，可以对存放在存储介质上的数据进行加密，也可以对在网络中传输的数据进行加密。加密技术在外包数据管理中的应用越来越引起人们的关注，在这样的应用背景下，需要解决对加密数据的操作问题。例如，如何对加密过的数据进行查询。

数据的完整性主要由访问控制机制和语义完整性约束来实现。每当一个主体要对某些数据进行修改时，访问控制机制就验证该主体是否拥有修改相应数据的权限，而语义完整性子系统则验证主体对数据所做的更新在语义上是否正确。语义正确性通过一组条件（即谓词）针对数据库的状态进行验证。为了能够检测数据是否可能被篡改，可以对数据进行数字签名。

数据的可用性主要由数据恢复子系统和并发控制机制来实现，就算软硬件出故障，或者多个并发程序对数据进行访问，它们也要确保数据是可以获得的，而且是正确的。数据的可用性，特别是 Web 上的数据的可用性，可以通过使用反拒绝服务攻击（DoS 攻击）技术来得到进一步增强，基于机器学习的反 DoS 攻击技术可以派上用场。

6.1.2　数据库安全的典型研究课题

通过 6.1.1 节的学习可以知道，访问控制机制是实现数据库系统安全性的重要机制。与操作系统的访问控制类似，数据库系统的访问控制也分为自主访问控制和强制访问控制。自主访问控制机制是数据库管理系统中最基本的也是应用最广泛的访问控制机制，关系数据模型是数据库系统中最基本和应用最广泛的数据模型，自然，关系数据库管理系统（RDBMS，Relational Database Management System）的自主访问控制机制也具有最基本和最广泛的意义。下面就从关系数据库管理系统的自主访问控制开始，了解数据库系统安全性的知识。

1．关系数据库自主访问控制

关系数据库系统的访问控制模型与操作系统的访问控制模型有不同之处。首先，关系数据库系统的访问控制模型应该用逻辑数据模型来表示，对关系数据库的授权应该用关系、关系属性和记录来表示。其次，在关系数据库系统中，除了基于名称的访问控制以外，还要支持基于内容的访问控制。在基于名称的访问控制中，通过指明客体的名称来实施对客体的保护。在基于内容的访问控制中，系统可以根据数据项的内容决定是否允许对数据项进行访问。借助 SQL 这样的描述性查询语言，在关系数据库系统中，很容易建立基于内容的访问控制模型。通常，这些访问控制模型是以数据内容的筛选条件的描述为基础的。

在关系数据库系统自主访问控制模型的发展中，System R 提供了一个经典的访问控制模型，对商用关系数据库管理系统访问控制模型产生了重大影响。System R 的访问控制模型具有以下特性：

（1）非集中式的授权管理；

（2）授权的动态发放与回收；

（3）通过视图支持基于内容的授权。

System R 的访问控制模型中的授权的发放与回收命令，即 GRANT 与 REVOKE 命令，后来得到了 SQL 标准的采纳。在 System R 的基本模型的基础上，关系数据库系统的自主访问控制模型不断得到拓展，新的特性不断丰富，典型的特性有：

（1）否定式授权；

（2）基于角色和基于任务的授权；

（3）基于时态的授权；

（4）环境敏感的授权。

自主访问控制模型对于实现数据库系统的访问控制具有广泛的意义，但是，它们存在一定的不足。例如，一旦主体获得授权访问到了信息，自主访问控制模型就无法对这些信息的传播和使用进行任何控制。

2．关系数据库强制访问控制

针对数据库系统自主访问控制存在的不足，人们希望通过强制访问控制和多级安全数据库系统来解决相应的问题。传统的数据库系统强制访问控制主要指基于贝尔-拉普杜拉模型的访问控制，贝尔-拉普杜拉模型是经典的多级安全模型，实现该模型的数据库系统属于多级安全数据库系统。

定义 6.1　多级安全数据库系统指以信息的等级划分为基础对数据进行访问控制的数据库系统。

多级安全数据库系统根据信息的敏感程度为信息确定安全等级，信息的安全等级越高，表示信息的敏感级别越高。同时，根据用户在工作中应该涉及的信息的敏感程度为用户分配安全等级，用户的安全等级越高，表示用户可以访问的信息的敏感级别越高；每当用户要对信息进行访问时，系统的安全机制根据用户的安全等级和信息的安全等级确定访问是否允许进行。

在多级关系数据库系统中，从理论上说，可以基于关系、关系属性或记录建立安全等级，在实际的系统开发中，切实可行的有效的方法是基于记录的安全等级进行访问控制。在记录级的多级安全关系数据库系统中，一张表中的不同记录可能具有不同的安全等级，因此，安全等级实际上把一张表分割成多张相互隔离的子表，不同子表中的记录具有不同的安全等级，拥有不同安全等级的用户实际上可以访问的是不同安全等级的子表中的记录数据。

在多级安全数据库系统中，需要考虑隐蔽信道（Covert Channel）的处理问题。迄今，国际上并没有关于隐蔽信道的统一定义。通俗的定义如下。

定义 6.2　如果存在某个渠道，原本是不该用于传递信息的，实际上却能被用于传递信息，那么，这样的渠道就称为一个隐蔽信道。

通常，利用隐蔽信道进行的信息传递，表面上并不违反安全原则，可实际上是违反安全原则的。系统中的很多成分或特性都有可能被利用来建立隐蔽信道。例如，系统时钟、出错信息、特定文件名的存在、进程间通信机制、并发控制机制等。

多级安全数据库系统的并发控制机制，如果对运行在不同安全等级的事务进行同步，就会引入一个明显的隐蔽信道，因此，面向并发控制的隐蔽信道处理，在数据库系统安全研究中引起了广泛的关注。

例 6.3　设高安全等级的主体 S1 执行高安全等级的事务 T1，低安全等级的主体 S2 执行低安全等级的事务 T2，如果并发控制机制对 T1 和 T2 进行同步，那么，借助系统的同步行为，S2 可以感知到 S1 的存在，无形中，这等同于高安全等级的主体 S1 在隐蔽地向低安全等级的主体 S2 传递信息，这就是隐蔽信道的一个示例。

多级安全关系数据库系统的多实例（Polyinstantiation）思想的提出与隐蔽信道的处理有关。

定义 6.3　在多级安全数据库系统中，多实例指的是在一张表中有多个主键相同的记录存在，这些记录的安全等级不同。

例 6.4　设在一张员工信息表 EMP 中，name 字段是主键，表中有多个 name 为“TOM”的记录存在，这些 name 为“TOM”的记录的安全等级互不相同，这就是一种多实例的情况。

为什么需要多实例呢？看一看下面的例子。

例 6.5　设安全等级为 L1 的高安全等级主体 S1 已经在表 EMP 中插入了一条主键字段 name 为“TOM”的记录，该记录的安全等级为 L1。

现在，安全等级为 L2 的低安全等级主体 S2 并不知道 EMP 中已有 name 为“TOM”的记录，它也要向表 EMP 中插入一条 name 为“TOM”的记录。

这时，如果系统因为表 EMP 中已有 name 为“TOM”的记录而不允许 S2 执行该插入操作，那么，S2 就能感知到 EMP 中已经有 name 为“TOM”的高安全等级的记录存在。借助这个渠道，S1 可以隐蔽地向 S2 传递信息，建立隐蔽信道。

为了防止产生这样的隐蔽信道，系统允许 S2 执行该插入操作，插入一条 name 为“TOM”且安全等级为 L2 的记录。因此，表 EMP 中有两条主键 name 均为“TOM”而安全等级分别为 L1 和 L2 的记录，产生了多实例。

这个例子告诉我们，实现多实例机制的重要原因是防止隐蔽信道的产生。

3．数据库安全技术的拓展

数据库技术的发展和应用技术的发展都要求我们在关系数据库管理系统安全模型的基础上进一步拓展数据库系统的安全技术。

例如，面向对象数据库管理系统需要建立面向对象数据库管理系统的安全模型，对象关系数据库管理系统需要建立对象关系数据库管理系统的安全模型。

新的数据管理系统和数据应用形态多样，如基于 XML 的 Web 数据、多媒体数据、数字图书馆、数据仓库系统、工作流系统等。其中的数据模型常常比关系数据模型丰富得多，典型地，它们蕴涵着语义建模（Semantic Modeling）的思想。

在新的应用背景下，系统需要保护的对象也增添了新的意义。不仅是数据本身需要保护，数据库模式也可能包含敏感信息，也不能允许主体随便地对它们进行访问。因此，也需要为数据库模式建立相应的访问控制策略。更进一步地看，访问控制策略本身也可能包含敏感信息，也需得到保护。也就是说，我们很可能需要建立一个访问控制策略，用于保护另一个访问控制策略。

而且，在新的应用背景下，系统需要处理多媒体数据，这给数据内容的自动解释带来了很大的困难。同时，系统还要向系统外部的用户提供服务，这些用户可能是通过 Web 接口来访问系统的，这也给系统的访问控制带来了新的挑战。

总而言之，需要对面向关系数据库管理系统的访问控制模型进行全方位的拓展，以支持新的建模思想，支持面向多媒体数据的基于内容的访问控制，支持面向外部用户的基于凭证（Credential-based）的访问控制。XML 的综合访问控制模型的开发就是一个典型的拓展性工作。

除了需要对传统的访问控制模型进行扩展外，新的数据应用背景也催生了新的数据库系统安全研究方向。例如，面向数据库外包的加密数据的查询处理技术就是一个重要的研究方向，其内容包括保序（Order-preserving）加密等特别的加密技术。支持隐私保护的（Privacy-preserving）数据库技术是另一个重要的研究方向，其内容包括支持隐私保护的数据挖掘、支持隐私保护的信息检索、支持隐私保护的数据库系统等。

以下内容也是与数据库系统安全性相关的重要研究课题：推理控制与统计数据库的安全性、GIS 数据的安全性、信息网格体系结构的安全性、传感数据的安全性、Web 服务和语义 Web 的私密性与安全性等。

6.2　关系数据库自主访问授权

在关系数据库管理系统的自主访问控制策略中，根据主体的标识和授权的规则控制主体对客体的访问。一个自主访问控制策略的自主性体现在它允许主体自主地把访问客体所需要的权限授予给其他的主体。

定义 6.4　一个授权可以通过以下具有一般性的式子来描述：

(S, O, A, P)

其中，S、O、A、P 分别表示主体、客体、访问类型和谓词。该授权所表达的意思是：当谓词 P 为真时，主体 S 有权对客体 O 进行 A 类型的访问。通常，A 可以表示在 O 上的查询、插入、更新或删除操作，等等。

如果使 P 取空值，则可以得到授权的简单形式：

(S, O, A)

它表示主体 S 有权对客体 O 进行 A 类型的访问。如果设 S 为用户 TOM，O 为表 EMP，A 为 SELECT，则以上授权表示用户 TOM 具有在表 EMP 上执行 SELECT 操作的权限。

6.2.1　授权的发放与回收

关系数据库管理系统自主访问控制的重要内容是授权的管理，系统中的授权管理指的是系统发放授权和回收授权的功能，该功能负责在访问控制机制中建立授权和撤销授权。

定义 6.5　建立授权的一般操作可以表示为以下形式，它表示把客体 O 上的 A 访问权授权给主体 S：

GRANT (S, O, A)

定义 6.6　撤销授权的一般操作可以表示为以下形式，它表示撤销主体 S 在客体 O 上的 A 访问权：

REVOKE (S, O, A)

常见的授权管理策略有集中式授权管理和基于属主的授权管理。

定义 6.7　集中式授权管理是这样的一种授权管理方式：它只允许一些拥有特权的主体发放和回收授权。

定义 6.8　基于属主的授权管理是这样的一种授权管理方式：它由客体的属主来负责发放和回收任意主体对相应客体的访问授权。

通常，基于属主的授权管理提供对委托授权的支持。

定义 6.9　委托授权指的是一个主体把对客体访问授权的发放和回收权传递给另一个主体，使得另一个主体能够发放和回收对相应客体的授权。

例 6.6　设 S1、S2 和 S3 是任意的主体，O1 是任意的客体，且 S1 是 O1 的属主，那么，在基于属主的授权管理策略中，S1 负责对 O1 的授权进行管理，它可以授权 S2 访问 O1，也可以撤销 S2 对 O1 的访问权，同时，S1 可以委托 S2 对 O1 进行授权管理，这样，S2 可以授权 S3 访问 O1，也可以撤销 S3 对 O1 的访问权。S2 还可以进一步委托 S3 对 O1 进行授权管理，依此类推，不断持续下去。这就是委托授权管理。

显然，委托授权管理是一种非集中式的授权管理，因而，基于属主的授权管理是非集中式的授权管理。

1．授权的发放与回收方法

利用 SQL 这种描述性语言，很容易实现关系数据库管理系统中基于属主的访问授权管理模型。

例 6.7　用户 SCOTT 是表 EMP 的属主，SCOTT 可以执行以下操作，给用户 BOB 发放在 EMP 上的查询授权，使 BOB 可以查询 EMP 中的记录。

GRANT SELECT ON emp TO bob

例 6.8　用户 SCOTT 是表 EMP 的属主，SCOTT 可以执行以下操作，给用户 BOB 发放在 EMP 上的插入授权，使 BOB 可以向 EMP 中插入记录。

```
GRANT INSERT ON emp TO bob
```

例 6.9　用户 SCOTT 是表 EMP 的属主，SCOTT 可以执行以下操作，给用户 BOB 发放在 EMP 上的更新授权，使 BOB 可以更新 EMP 中的记录。

```
GRANT UPDATE ON emp TO bob
```

例 6.10　用户 SCOTT 是表 EMP 的属主，SCOTT 可以执行以下操作，给用户 BOB 发放在 EMP 上的删除授权，使 BOB 可以删除 EMP 中的记录。

```
GRANT DELETE ON emp TO bob
```

例 6.11　用户 SCOTT 是表 EMP 的属主，SCOTT 可以执行以下操作，回收用户 BOB 在 EMP 上的查询授权，使 BOB 不能查询 EMP 中的记录。

```
REVOKE SELECT ON emp FROM bob
```

例 6.12　用户 SCOTT 是表 EMP 的属主，SCOTT 可以执行以下操作，给用户 BOB 发放在 EMP 上的查询授权，使 BOB 可以查询 EMP 中的记录，并且，把 EMP 上的查询授权委托给 BOB，使 BOB 可以给其他用户发放在 EMP 上的查询授权，或者回收其他用户在 EMP 上的查询授权。

```
GRANT SELECT ON emp TO bob WITH GRANT OPTION
```

例 6.13　用户 BOB 已经获得在表 EMP 上的查询授权的委托，BOB 可以执行以下操作，给用户 ALICE 发放在 EMP 上的查询授权，使 ALICE 可以查询 EMP 中的记录。

```
GRANT SELECT ON emp TO alice
```

例 6.14　用户 SCOTT 是表 EMP 的属主，SCOTT 可以执行以下操作，回收用户 BOB 在 EMP 上的查询、插入、更新和删除授权。

```
REVOKE SELECT, INSERT, UPDATE, DELETE
ON emp FROM bob
```

例 6.15　用户 SCOTT 是表 EMP 的属主，SCOTT 执行以下操作，给用户 BOB 发放在 EMP 上的查询、插入、更新和删除授权，并把 EMP 上的查询、插入、更新和删除授权委托给 BOB。

```
GRANT SELECT, INSERT, UPDATE, DELETE
    ON emp TO bob
        WITH GRANT OPTION
```

上面的几个例子直观地告诉我们，授权的发放和回收操作是针对某张表的某种访问类型进行工作的。给用户发放了在某张表上的某种访问类型的授权后，该用户就能够对相应的表进行相应的访问了。例如，例 6.7 给用户 BOB 发放了在表 EMP 上的查询授权后，BOB 就能够对 EMP 进行 SELECT 访问了。回收了用户在某张表上的某种访问类型

的授权后，该用户就不能对相应的表进行相应的访问了。例如，例 6.11 回收了用户 BOB 在 EMP 上的查询授权后，BOB 就不能对 EMP 进行 SELECT 访问了。

2．委托管理下的授权回收

授权的委托管理会给授权回收操作的语义带来一些新的问题。

如果把某张表上的某个授权的管理权委托给了某个主体，当要回收该主体拥有的相应授权时，该主体很有可能已经把相应的授权发放给了其他的主体，这时，只回收该主体的授权也许是不够的。

例如，例 6.12 中把表 EMP 上的 SELECT 授权的管理权委托给了用户 BOB，例 6.14 要回收 BOB 在 EMP 上的 SELECT 授权，可是，在此之前，在例 6.13 中，BOB 已经把在 EMP 上的 SELECT 授权发放给了用户 ALICE。

从本质上看，ALICE 之所以拥有在 EMP 上的 SELECT 授权，完全是因为把 EMP 上的 SELECT 授权的管理权委托给 BOB 的缘故，所以，回收 BOB 在 EMP 上的 SELECT 授权时，不能不考虑 ALICE 在 EMP 上的 SELECT 授权。

因而，在委托授权管理的环境下，授权回收操作的语义应该定义如下。

定义 6.10　主体 S1 回收主体 S2 的 A 授权的操作是成功的，当且仅当，回收操作完成后，只有在 S1 把 A 授权的管理权委托给 S2 前存在的那些授权才是有效的。

以上定义的意思是，S1 回收 S2 的 A 授权后，系统中的授权情况应该与 S1 把 A 授权的管理权委托给 S2 前相同，就好像 S1 从来没有把 A 授权的管理权委托给 S2 一样。

回顾上面的例子，回收 BOB 在 EMP 上的 SELECT 授权后，系统中的授权情况应该与把 EMP 上的 SELECT 授权的管理权委托给用户 BOB 前相同，此时，ALICE 不应该拥有在 EMP 上的 SELECT 授权，可见，在回收 BOB 在 EMP 上的 SELECT 授权时，也应该回收 ALICE 在 EMP 上的 SELECT 授权。

为了满足定义 6.10 中的条件，每当回收某个主体在某张表上的某个授权时，需要进行递归的授权回收操作，以便把因该授权的发放而得以建立的在该表上的所有授权全部撤销。

6.2.2　否定式授权

前面谈到的数据库访问授权都是肯定式授权，也就是说，当给某个主体授予某种权限时，就表示允许该主体对某个客体进行某种访问，相反，当不给主体授权时，就表示不允许该主体对客体进行访问。例如，给用户 BOB 授予在表 EMP 上的 SELECT 权限，就表示允许 BOB 查询 EMP 中的记录，相反，不给 BOB 授予在 EMP 上的 SELECT 权限，就表示不允许 BOB 查询 EMP 中的记录。

在只有肯定式授权的访问控制策略下，当一个主体试图对一个客体进行访问时，如

果在系统中能找到相应的访问授权，则访问可以进行，如果在系统中找不到相应的访问授权，则访问就被拒绝。也就是说，没有授权等同于禁止访问。

采取这样的方法来实现禁止访问的需求存在一定的不足。因为，如果想禁止一个主体对一个客体进行访问，唯一的办法就是不给该主体授予访问对应客体的权限。可是，现在不给主体授权，并不能保证该主体将来都不会获得所需的权限。毕竟，所有拥有某个客体的授权管理权的主体，都能够给其他主体授予访问该客体的权限。

例 6.16　用户 BOB 拥有表 EMP 的授权管理权，BOB 要禁止用户 TOM 查询 EMP 中的记录，BOB 的办法就是不给 TOM 发放 EMP 上的 SELECT 授权，此刻，TOM 确实无法查询 EMP 中的记录。用户 ALICE 也拥有 EMP 的授权管理权，她不知道其他用户禁止 TOM 查询 EMP，因此，她给 TOM 发放了 EMP 上的 SELECT 授权，这时，TOM 便能够对 EMP 进行查询操作。至此，BOB 无法达到禁止 TOM 查询 EMP 中的记录的目的。

引入否定式授权可以解决访问控制中的这一不足。

前面我们用 A 来描述 A 类型的授权，这是肯定式授权，即给主体 S 发放客体 O 上的一个 A 授权，其意义是授权主体 S 对客体 O 进行 A 类型的访问。

定义 6.11　设 A 为任意的授权，用 NOA 来标记与 A 授权相关的另一个授权，它的含义是禁止 A 授权，即，给主体 S 发放客体 O 上的一个 NOA 授权的含义是禁止主体 S 对客体 O 进行需要 A 授权才能进行的访问。NOA 称为相对于授权 A 的一个否定式授权，发放 NOA 授权的过程称为进行否定式授权。

例 6.17　设 NODELETE 是与 DELETE 对应的否定式授权，那么，给用户 BOB 发放表 EMP 上的 NODELETE 授权，就表示禁止 BOB 在 EMP 上执行 DELETE 操作，从而禁止 BOB 删除 EMP 中的记录。

在同时支持肯定式授权和否定式授权的系统中，出现授权冲突是避免不了的。如果主体 S 既获得了在客体 O 上的肯定式 A 授权，也获得了在客体 O 上的否定式 NOA 授权，冲突就发生了。在这种情况下，是允许 S 对 O 进行 A 操作还是禁止 S 对 O 进行 A 操作呢？我们采用以下的否定优先原则解答这个问题。

规则 6.1（否定优先原则）　如果一个主体既拥有在某个客体上进行某种操作的肯定式授权，也拥有在该客体上进行同样操作的否定式授权，那么，否定式授权发挥作用，肯定式授权不起作用。

注意：只要主体 S 同时拥有在客体 O 上的肯定式 A 授权和否定式 NOA 授权，那么，起控制作用的一定是 NOA 授权，不管 A 授权和 NOA 授权哪个先发放，哪个后发放，系统都禁止 S 对 O 进行 A 类型的访问。

例 6.18　设用户 BOB 和 ALICE 都拥有表 EMP 的授权管理权，BOB 在 T_1 时刻给 TOM 发放了 EMP 上的否定式 NOSELECT 授权，在 T_1 之后的 T_2 时刻，ALICE 给 TOM 发放了 EMP 上的肯定式 SELECT 授权。请问：TOM 在 T_2 时刻之后能否查询 EMP 中的记录？

解答：TOM 同时拥有在 EMP 上进行 SELECT 操作的肯定式授权和否定式授权，根据否定优先原则，系统禁止 TOM 对 EMP 进行 SELECT 操作，授权的先后顺序对此没有影响，所以，TOM 不能查询 EMP 中的记录。

例 6.18 解决了例 6.16 中存在的问题。

否定式授权也可应用于在普遍授权的环境中临时屏蔽某个主体可能获得的肯定式授权，以创造一些例外场景。

例如，如果想授权某个小组中除了某个组员以外的其他所有组员对某张表进行访问，那么，可以给该小组发放一个在给定表上的指定访问类型的肯定式授权，然后，给该组员发放一个在同一张表上的相同访问类型的否定式授权。

例如，前面提到的小组、组员、表和访问类型分别是 G1、U1、O1 和 A1，就是想让 G1 中的所有用户都能对 O1 进行 A1 操作，但 U1 是个例外，要禁止他的操作，做法就是给 G1 发放 O1 上的 A1 授权，给 U1 发放 A1 上的 NOA1 授权。

这里涉及小组和组员的授权问题，而且授权是冲突的，一个是肯定式授权，一个是否定式授权。访问控制应该以哪个授权为准呢？采用以下的个体优先原则处理这个问题。

规则 6.2（个体优先原则）　如果某个小组及其组中的某个用户都拥有同一个客体上的同一种访问类型的授权，并且，两者所拥有的授权是冲突的，那么，用户拥有的授权发挥作用，小组拥有的授权不起作用。

这个规则确定了小组与组员间的授权冲突处理办法。当小组与组员间存在授权冲突时，总是组员所拥有的授权覆盖了小组所拥有的授权，不管组员所拥有的授权是否定式授权还是肯定式授权。

例 6.19　设用户 BOB 拥有表 EMP 的授权管理权，用户 TOM 是小组 G1 中的成员，BOB 给 G1 发放了 EMP 上的 SELECT 授权。请问：TOM 能否查询 EMP 中的记录？

解答：TOM 是 G1 中的成员，G1 拥有 EMP 上的 SELECT 授权，因而 TOM 也拥有 EMP 上的 SELECT 授权，所以 TOM 可以查询 EMP 中的记录。

例 6.20　设用户 BOB 拥有表 EMP 的授权管理权，用户 TOM 是小组 G1 中的成员，BOB 给 G1 发放了 EMP 上的肯定式 SELECT 授权，给 TOM 发放了 EMP 上的否定式 NOSELECT 授权。请问：TOM 能否查询 EMP 中的记录？

解答：G1 拥有 EMP 上的 SELECT 授权，TOM 拥有 EMP 上的 NOSELECT 授权，

这两个授权是冲突的，根据个体优先原则，TOM 拥有的授权发挥作用，所以 TOM 不能查询 EMP 中的记录。

例 6.21　设用户 BOB 拥有表 EMP 的授权管理权，用户 TOM 是小组 G1 中的成员，BOB 给 G1 发放了 EMP 上的否定式 NOSELECT 授权，给 TOM 发放了 EMP 上的肯定式 SELECT 授权。请问：TOM 能否查询 EMP 中的记录？

解答：G1 拥有 EMP 上的 NOSELECT 授权，TOM 拥有 EMP 上的 SELECT 授权，这两个授权是冲突的，根据个体优先原则，TOM 拥有的授权发挥作用，所以 TOM 可以查询 EMP 中的记录。

在例 6.19 中，并没有给用户 TOM 授权，但给组 G1 授予了查询表 EMP 的权限，由于 TOM 是 G1 中的成员，所以 TOM 也拥有了查询 EMP 的权限。

在例 6.20 中，给 G1 授予了查询 EMP 的权限，给 TOM 授了禁止查询 EMP 的权限，结果是 TOM 不能查询 EMP，否定式授权覆盖了肯定式授权。

在例 6.21 中，给 G1 授予了禁止查询 EMP 的权限，给 TOM 授予了查询 EMP 的权限，结果是 TOM 可以查询 EMP，肯定式授权覆盖了否定式授权。

结合个体优先原则和否定优先原则，授权冲突问题的整体解决方案是：在小组与用户的授权之间，用户的授权覆盖小组的授权；在用户自己的授权之间，或在小组自己的授权之间，否定式授权覆盖肯定式授权。

作为安全数据库管理系统的一个实例，SeaView 系统支持否定式授权，它只提供一个否定式权限，即 NULL 权限。NULL 权限禁止主体对客体进行任何类型的访问。

例 6.22　设在 SeaView 系统中，用户 BOB 和 ALICE 拥有表 EMP 的授权管理权，在 T_1 时刻，BOB 给用户 TOM 发放了 EMP 上的 SELECT、INSERT、UPDATE 和 DELETE 授权，在 T_2 时刻，ALICE 给 TOM 发放了 EMP 上的 NULL 授权。请问：

（1）在 T_1 时刻之后 T_2 时刻之前，TOM 对 EMP 拥有什么访问权限？

（2）在 T_2 时刻之后，TOM 对 EMP 拥有什么访问权限？

解答：（1）因为在 T_1 时刻，BOB 授权 TOM 对 EMP 进行查询、插入、更新和删除操作，所以，在 T_1 之后 T_2 之前，TOM 对 EMP 拥有查询、插入、更新和删除访问权限。

（2）在 T_2 时刻，ALICE 给 TOM 授予了 NULL 权限，这是禁止进行任何类型的访问的否定式权限，所以，在 T_2 时刻之后，TOM 被禁止对 EMP 进行任何访问。

SeaView 系统根据以下规则处理授权的冲突问题：

（1）直接发放给用户的授权覆盖发放给用户所在的小组的授权。

（2）发放给一个主体的 NULL 授权覆盖发放给该主体的所有其他授权。

SeaView 系统处理授权冲突的这两个规则与规则 6.1（否定优先原则）及规则 6.2（个体优先原则）是一致的。

例 6.23　设在 SeaView 系统中，用户 BOB 拥有表 EMP 的授权管理权，用户 TOM 是小组 G1 中的成员，BOB 给 G1 发放了 EMP 上的 NULL 授权，给 TOM 发放了 EMP 上的 SELECT 授权。请问：TOM 能否查询 EMP 中的记录？

解答：虽然 TOM 在 G1 中，NULL 授权禁止 G1 中的用户对 EMP 进行任何类型的访问，但是 TOM 直接获得了对 EMP 的 SELECT 访问权，根据 SeaView 系统的授权冲突处理规则，直接发放给用户 TOM 的授权覆盖发放给 TOM 所在的小组 G1 的授权，所以 TOM 可以查询 EMP 中的记录。

6.2.3　可选的授权回收方式

在允许进行授权的委托管理的情况下，回收授权时，需要考虑授权传递的连带效果。定义 6.10 描述了这种情形下授权回收操作的相关语义。委托管理下的授权传递可以借助图 6.1 回顾并进一步说明这方面的问题。

图 6.1 描绘了在委托管理下授权传递的一个场景，其中，S1、S2、S3 和 S4 表示主体，箭头表示授权的发放。最初，S1 给 S2 发放授权 A，并把授权 A 的管理权委托给 S2；后来，S2 给 S3 发放授权 A，并把授权 A 的管理权委托给 S3；最后，S3 把授权 A 发放给 S4。在这个图例中，S1 只把授权 A 发放给了 S2，而实际上，授权 A 传递到了 S4，这是 S2 和 S3 获得授权 A 的委托管理权后发放了授权 A 的结果。

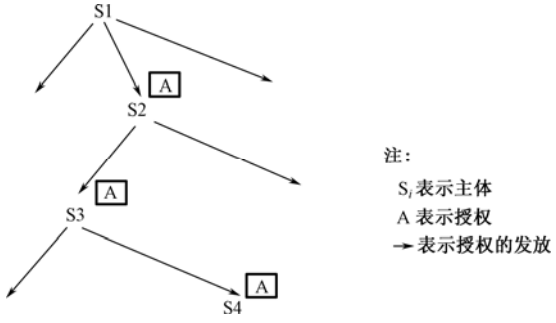


图 6.1　委托管理下的授权传递

按照定义 6.10 的要求，当 S1 要从 S2 回收授权 A 时，会触发递归的授权回收操作，系统将相继把 S4、S3 和 S2 中的授权 A 都回收，使 S4、S3 和 S2 都失去授权 A，因为该授权是由 S1 发放的，而现在 S1 要回收它，尽管 S3 和 S4 中的授权 A 不是由 S1 直接发放的。

授权的递归回收反映了实际应用中的一类需求，同时也要看到，并非在所有的应用中都应该实行授权的递归回收。

在某些应用中，用户所拥有的授权与其工作岗位和职责有关，当用户调离工作岗位时，系统需要回收其拥有的相应授权，但没有必要回收工作岗位不变的其他用户拥有的

授权，哪怕这些用户拥有的授权是由调离岗位的那个用户发放的。

例 6.24　结合图 6.1，设 S1 是某大学的校长，S2 是该大学某学院的院长，S3 是该学院某系的主任，S4 是该系的一位教师；校长 S1 根据工作需要把授权 A 发放给院长 S2，并委托他管理该授权，院长 S2 根据工作需要把授权 A 发放给系主任 S3，并委托他管理该授权，系主任 S3 根据工作需要把授权 A 发放给教师 S4。当 S2 调离院长岗位时，校长 S1 要回收他拥有的授权 A，请问：系统是否应该同时回收 S3 和 S4 拥有的授权 A？

解答：虽然 S3、S4 所拥有的授权 A 分别是由 S2、S3 发放的，实际上，S3 和 S4 之所以获得授权 A，是 S1 把授权 A 发放给 S2 并委托他管理所形成的，但是，现在 S1 要回收 S2 拥有的授权 A，是因为他调离了院长岗位，可是，S3 依然是系主任，S4 依然是该系的老师，他们拥有的授权 A 是根据工作需要发放的，由于他们的工作没有变化，所以系统不应该回收他们拥有的授权 A。

考虑到实际应用中两类不同的授权回收需求，关系数据库管理系统可以相应地提供两种类型的授权回收功能，即递归式授权回收功能和非递归式授权回收功能。这里所说的递归式回收也称为级联（Cascade）回收，自然，非递归式回收也称为非级联回收。

定义 6.12　递归式授权回收指这样的一种授权回收过程：它既要回收指定主体拥有的指定授权，也要回收由该主体直接和间接传递给其他主体的该授权。

定义 6.13　非递归式授权回收指这样的一种授权回收过程：它只需回收指定主体拥有的指定授权，无须回收由该主体直接或间接传递给其他主体的该授权。

例 6.25　在图 6.1 中，主体 S1 通过递归式授权回收操作从主体 S2 中回收授权 A，请问：回收操作完成后，主体 S3 和 S4 是否拥有授权 A？

解答：S3 拥有的授权 A 是通过 S2 的直接传递获得的，S4 拥有的授权 A 是通过 S2 的间接传递获得的，执行递归式授权回收操作时，S2 直接和间接传递的授权 A 都被回收，所以，回收操作完成后，S3 和 S4 不再拥有授权 A。

例 6.25　在图 6.1 中，主体 S1 通过非递归式授权回收操作从主体 S2 中回收授权 A，请问：回收操作完成后，主体 S3 和 S4 是否拥有授权 A？

解答：执行非递归式授权回收操作时，系统只回收 S2 拥有的授权 A，不回收 S2 传递的授权，所以，回收操作完成后，S3 和 S4 依然拥有授权 A。这时，系统可以把 S3 拥有的授权 A 作为 S1 发放的授权对待。

这里介绍的递归式授权回收和非递归式授权回收是两种可以实现的授权回收方式。不同的系统可能实现不同的授权回收支持。

在 SQL 中，授权回收操作 REVOKE 中的 CASCADE 和 RESTRICT 选项与递归式授权回收相关。CASCADE 选项表示要实施递归回收。RESTRICT 选项表示要禁止实施递归回收。当系统中存在授权传递情形时，RESTRICT 选项禁止回收操作 REVOKE 的执行。

例 6.26　在图 6.1 中，设授权 A 表示表 EMP 上的 SELECT 授权，主体 S1 执行以下操作会得到什么结果？

REVOKE SELECT ON emp FROM s1 RESTRICT

解答：因为 S2 把授权 A 传递给了 S3 和 S4，系统中存在授权传递情形，RESTRICT 选项禁止 REVOKE 操作执行，所以授权回收不成功。

例 6.27　在图 6.1 中，设授权 A 表示表 EMP 上的 SELECT 授权，主体 S1 执行以下操作会得到什么结果？

REVOKE SELECT ON emp FROM s1 CASCADE

解答：CASCADE 选项促使系统执行递归回收操作，所以，S2、S3 和 S4 拥有的授权 A 全部被回收。

在大多数关系数据库管理系统中，在默认情况下，REVOKE 操作执行的是递归式授权回收，所以，不指定 CASCADE 选项与指定 CASCADE 选项的效果相同。

一个主体执行的授权回收操作只能回收自己发放的授权，对其他主体发放的授权没有影响。如果两个不同的主体给第三个主体发放了同一个授权，那么，在这两个主体中的其中一个从第三个主体回收了该授权后，第三个主体仍然拥有该授权。多方授权与授权传递如图 6.2 所示，主体 S0 和 S1 都给主体 S2 发放了授权 A；S1 可以回收它给 S2 发放的授权 A，但它回收不了 S0 给 S2 发放的授权 A。

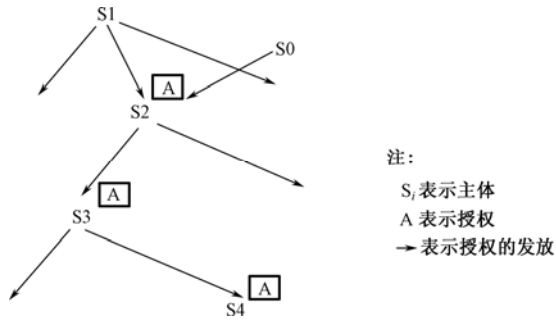


图 6.2　多方授权与授权传递

例 6.28　在图 6.2 中，设授权 A 表示表 EMP 上的 SELECT 授权，主体 S0 和 S1 都给 S2 发放了授权 A，主体 S1 执行以下操作：

REVOKE SELECT ON emp FROM s1 CASCADE

请问：操作完成后，S2、S3 和 S4 是否拥有授权 A？

解答：就算 S1 撤销了它发放给 S2 的授权 A，S2 也还拥有 S0 发放给它的授权 A，

所以 S2 不会失去授权 A，因而，S3 和 S4 拥有的授权 A 不受影响，所以，REVOKE 操作完成后，S2、S3 和 S4 依然拥有授权 A。

6.2.4　授权的时效性

在前面讨论的有关授权的所有情形中，某个主体一旦获得了某个授权，该授权就永远有效，除非该授权被明确地撤销。

例 6.29　以下授权发放操作成功执行后，主体 S 就获得了在客体 O 上的 A 访问权限，如果没有人撤销该授权，S 将永远有权在 O 上执行 A 操作。

GRANT (S, O, A)

例 6.30　在例 6.29 的成功授权之后，如果要使主体 S 丧失在客体 O 上的 A 访问权限，可以执行以下撤销授权的操作。

REVOKE (S, O, A)

当然，例 6.30 中的撤销授权的操作既可以是由某主体发出的，也可以是由某递归式授权回收操作触发的，但不管是哪种情况，归根到底都是因为主体发出了撤销授权的指令。

在实际应用中，有时需要指明授权的时效性，限定授权只能在具体的时间段内有效，在该时间段以外，授权无效。

定义 6.14　为了表示授权的时效特性，可以引入一个时效元素 T，用以下具有一般性的式子

（S, O, A, T）

表示主体 S 在时段 T 内拥有在客体 O 上执行 A 操作的权限，在时段 T 之外，S 没有在 O 上执行 A 操作的权限。

给授权增加了时效元素 T 之后，主体不需要执行撤销授权的操作，只要在时段 T 之外，系统就自动撤销相应的授权。

例 6.31　设用户 BOB 拥有表 EMP 的授权管理权，用户 ALICE 在 EMP 上没有任何访问权限，BOB 执行以下操作给 ALICE 发放授权：

GRANT (ALICE, EMP, SELECT, T)

如果用(t_1～t_2)表示 t_1～t_2 的时间段，t_1 和 t_2 是形如 yyyymmdd 的日期值，其中，yyyy 是表示年份的四位数，mm 是表示月份的两位数，dd 是表示日子的两位数，T 表示为

(20080715-20080930)

另设 BOB 执行以上授权操作的日期是 2008 年 7 月 5 日，请问：

（1）在 2008 年 7 月 6 日，ALICE 在 EMP 上拥有什么访问权限？

（2）在 2008 年 8 月 20 日，ALICE 在 EMP 上拥有什么访问权限？

（3）在 2008 年 10 月 1 日，ALICE 在 EMP 上拥有什么访问权限？

解答：T 的取值确定授权的有效期为 2008 年 7 月 15 日至 2008 年 9 月 30 日，所以：

（1）授权尚未生效，ALICE 在 EMP 上没有任何访问权限；

（2）授权生效，ALICE 拥有在 EMP 上的 SELECT 访问权限；

（3）授权已经失效，ALICE 在 EMP 上没有任何访问权限。

为了表示类似例 6.31 中的时间段的值，可以按照以下形式约定日期的表示方法。

约定 6.1　用一个形如 yyyymmdd 的八位数表示一个日期值，其中，yyyy 是表示年份的四位数，mm 是表示月份的两位数，dd 是表示日子的两位数。实际年份、月份或日子的位数不足时，用 0 补足。在年份位上，用****表示任意年份，在月份位上，用**表示任意月份，在日子位上，用**表示任意日子。

显然，根据以上约定，yyyy****、****mm**、******dd 分别单独表示年份、月份和日子。

约定 6.2　用(d_1, d_2, …)的形式表示时效元素 T 的值，其中，d_i 可以是一个日期值 t_j 或是一个形如 t_1～t_2 的日期范围值。

显然，根据以上约定，以下都是时效元素 T 的有效值：

(20080705)；

(20080705, 20080805)；

(20080715-20080930)；

(20080705, 20080715-20080930, 20081113)。

在考虑授权的时效性时，不但要考虑时间段问题，有时也需要考虑周期性问题。例如，使某个授权在每月的 1～20 日内有效。

例 6.32　设用户 BOB 拥有表 EMP 的授权管理权，用户 ALICE 在 EMP 上没有任何访问权限，BOB 执行以下操作给 ALICE 发放授权：

GRANT (ALICE, EMP, SELECT, (******01-******20))

请问：ALICE 何时拥有在 EMP 上的何种访问权？

解答：ALICE 在每月的 1～20 日拥有在 EMP 上的 SELECT 访问权。

例 6.33　设用户 BOB 拥有表 EMP 的授权管理权，用户 ALICE 在 EMP 上没有任何访问权限，为使 ALICE 在每年的第一和第四季度能够查询 EMP 中的信息，请问 BOB 需要执行怎样的授权操作？

解答：BOB 可以执行以下授权操作：

GRANT (ALICE, EMP, SELECT,
(****01**-****03**, ****10**-****12**))

星期是常用的周期性时效概念，为了利用星期来表示时效元素 T，可以建立以下有关星期的约定。

约定 6.3　用 W1、W2、W3、W4、W5、W6、W7 分别表示星期一、星期二、星期三、星期四、星期五、星期六、星期日，用(d_1, d_2, …)的形式表示时效元素 T 的值，其中，d_i 可以是一个星期值 t_j 或是一个形如 t_1～t_2 的星期范围值。

约定 6.3 给出的是星期型的时效元素值，约定 6.2 给出的是日期型的时效元素值。

例 6.34　用户 BOB 拥有表 EMP 的授权管理权，用户 ALICE 在 EMP 上没有任何访问权限，BOB 执行以下授权操作，授权 ALICE 在工作日内查询 EMP 中的信息：

GRANT (ALICE, EMP, SELECT, (W1-W5))

例 6.35　用户 BOB 拥有表 EMP 的授权管理权，用户 TOM 在 EMP 上没有任何访问权限，BOB 执行以下授权操作，授权 TOM 在星期六和星期日修改 EMP 中的信息：

GRANT (ALICE, EMP, UPDATE, (W6, W7))

为了更灵活地确定授权的时效元素值，可以用已有的时效元素值构造新的时效元素值，为此，可以建立以下约定。

约定 6.4　设 T1 和 T2 是两个时效元素值，可以通过这两个值的 AND 和 OR 运算，构造新的时效元素值，约定如下：

T1 AND T2　：等于 T1 时效与 T2 时效的相交时效的值；
T1 OR T2　：等于 T1 时效与 T2 时效的相并时效的值。

下面的例子进一步说明 AND 和 OR 运算的含义。

例 6.36　设：

T1 = (W1, W2)；
T2 = (W2, W3)；
T3 = (****03**, ****05**)；
T4 = (****05**, ****07**)。

则有：

T1 AND T2 = (W2)；
T1 OR T2 = (W1, W2, W3)；
T3 AND T4 = (****05**)；
T3 OR T4 = (****03**, ****05**, ****07**)；

T1 AND T3 =（3 月份的星期一和星期二，5 月份的星期一和星期二）。

例 6.37　用户 BOB 拥有表 EMP 的授权管理权，用户 ALICE 在 EMP 上没有任何访问权限，为使 ALICE 在每月 1～20 日的工作日内能够查询 EMP 中的信息，BOB 需要执行怎样的授权操作？

解答：BOB 可以执行以下授权操作：

```
GRANT (ALICE, EMP, SELECT,
    (******01-******20) AND (W1-W5))
```

本节介绍了在访问授权中考虑时效因素的基本思想和方法。注意：这里强调的是基本思想和基本方法，访问授权中要求表达的时效元素比这里讨论的要复杂得多，而且，目前大多数的关系数据库管理系统并没有提供拥有时效性的授权支持，我们给出的授权方法也不是 SQL 的授权方法，而仅是原理性的说明，但是，授权的时效性是现实应用中的一种需求，值得关注。

6.2.5　系统级的访问授权

前面给出的关于访问控制授权问题的讨论是紧紧围绕关系数据库系统表中数据的访问许可而展开的，其中所涉及的授权是数据库对象级的访问授权。表是关系数据库系统中的典型对象，除了表以外，视图和同义词等也是关系数据库系统中的重要对象，这些对象的访问授权与表的访问授权类似。

数据库数据需要保护，数据库模式也需要保护。对数据库模式的访问授权属于系统级的访问授权。以下是访问数据库模式所需要的典型授权。

（1）创建表的授权（CREATE TABLE）：允许创建表；

（2）删除表的授权（DROP TABLE）：允许删除表；

（3）增加字段的授权（ADD ATTRIBUTE）：允许给表增加字段；

（4）删除字段的授权（DELETE ATTRIBUTE）：允许删除表中字段。

以上这些授权是对数据库模式进行修改操作所需要的典型授权。一般而言，只有数据库模式的属主才能对数据库模式进行修改操作。但是，有的数据库管理系统也提供灵活的数据库模式访问控制授权。以下是授权用户修改数据库模式的一个例子。

例 6.38　下面的操作授权用户 BOB 创建表：

```
GRANT CREATE TABLE TO bob
```

下面的操作授权用户 ALICE 删除表：

```
GRANT DROP TABLE TO alice
```

获得了创建表的授权的用户，在创建新表时，如果想要引用其他表的主键（Primary Key）作为新表的属性（Attribute），换句话说，想要通过外部键（Foreign Key）来定义

新表的属性，还需要获得对相应主键的引用授权。下面是引用外部键定义表的属性的一个例子。

例 6.39　设客户表 CUSTOMERS 是一张已有的表，用户 ALICE 现在想要创建一张订单表 ORDERS，两张表的模式分别如下。

CUSTOMERS：customer_no, name, phone, address

ORDERS：order_no, customer_no, total

其中，客户号 customer_no 是 CUSTOMERS 的主键，ALICE 想要引用 CUSTOMERS 的这个主键来作为 ORDERS 中的属性，因此，ALICE 需要获得对 CUSTOMERS 中的 customer_no 的引用授权。

下面这个例子给用户 ALICE 发放一个她所需要的引用授权。

例 6.40　下面的操作授权用户 ALICE 引用表 CUSTOMERS 的主键 customer_no 定义新表的属性。

```
GRANT REFERENCES (customer_no) ON customers TO alice
```

获得了例 6.40 发放的 REFERENCES 授权后，ALICE 就可以创建例 6.39 中的 ORDERS 表了。

在数据库系统中，除了有存储应用信息的表以外，还有存储系统信息的表，如果把前者称为应用表的话，则后者就是系统表。系统表是执行系统管理任务所需要的表。系统级的访问授权也包括对系统表的访问控制授权。普通用户没有访问系统表的权限，要想访问系统表，必须获得相应的系统级的访问授权。

例 6.41　表 DBA_SYS_PRIVS 是一张存放系统授权信息的系统表，其中记录着哪个用户拥有哪些访问授权等方面的信息，数据库管理员 DBA 拥有对它的管理权，为使用户 ALICE 能够查询该系统表的信息，DBA 可以进行以下授权：

```
GRANT SELECT ON dba_sys_privs TO alice
```

获得该授权后，ALICE 可以执行以下操作查看系统的授权信息：

```
SELECT DISTINCT PRIVILEGE FROM dba_sys_privs
```

系统级的访问授权很多，除了上面提到的授权以外，下面这些也是非常典型的。

（1）创建触发器的授权（CREATE TRIGGER）：允许创建触发器；

（2）执行存储过程或函数的授权（EXECUTE）：允许执行存储过程或函数；

（3）建立会话的授权（CREATE SESSION）：允许登录到数据库中；

（4）改变会话的授权（ALTER SESSION）：允许改变会话环境的参数；

（5）创建用户的授权（CREATE USER）：允许创建用户；

（6）删除用户的授权（DROP USER）：允许删除用户；

（7）系统审计的授权（AUDIT SYSTEM）：允许进行系统审计；

（8）数据库维护的授权（ALTER DATABASE）：允许对数据库进行维护；

（9）系统维护的授权（ALTER SYSTEM）：允许对系统进行维护。

以上这些只是系统级的访问授权的一个缩影。我们并不打算对数据库系统中所有系统级的访问授权进行全面介绍，实际上，也不可能给出这样的介绍，因为不同的数据库管理系统还会提供具有自身特点的访问授权。但是，希望大家能够建立系统级的访问授权的基本概念，因为这是数据库系统安全中不可忽视的重要内容。

另外，整个数据库系统是以文件的形式存在于计算机系统之中的，要实现数据库系统的安全性，也必须保障这些文件的安全性，这是通过操作系统的安全机制来实现的。

6.3　基于视图的访问控制

6.2 节介绍了数据库中对象级的访问控制。对象级的访问控制以整个对象为单位来进行访问授权，在这种授权方式中，用户要么可以对整个对象进行访问，要么对整个对象都不能访问。当对象是一张表时，在这样的授权方式中，用户要么可以访问表中的所有内容，要么无法访问表中的任何内容。

6.3.1　基于内容的访问控制需求

现实应用常常要求系统根据实际需要允许用户访问表中的一部分内容，而不一定是表中的全部内容，也就是说，要求系统提供基于内容的访问控制。以下是一个典型的需求。

例 6.42　基于内容的访问控制需求：在对职员信息表实施的访问控制中，允许且只允许项目经理查看那些参加他所负责的项目的职员的信息。

根据以上访问控制策略，当一个用户想要查询职员信息表时，如果他是项目经理，那么，他所能查询到的结果取决于表中信息的内容。如果表中的记录信息对应的职员参加了该项目经理所负责的项目，那么，记录信息就出现在查询结果中，否则记录信息就不出现在查询结果中。

基于内容的访问控制是数据库管理系统的访问控制机制应该满足的一个重要需求。从本质上说，基于内容的访问控制要求根据数据的内容进行访问控制判定。

6.3.2　基于视图的读访问控制

关系数据库管理系统中的视图（View）机制可用于提供基于内容的访问控制支持。视图是能够展示数据库对象实例中的字段和记录的子集的动态窗口，它通过查询操作来确定所要展示的字段和记录的子集。

例 6.43　设属主对用户 SCOTT 的职员信息表 EMP 的定义为

```
EMP：ename, job, deptno, sal, sex, age, phone, …
```

各字段表示的分别是姓名、岗位、部门号、薪水、性别、年龄、电话、地址等。

如果访问控制策略要求只允许 TOM 等用户查询不超过 30 岁的普通职员的姓名、性别、部门号和电话信息，SCOTT 可以通过以下操作创建视图 YOUNG_CLERKS：

```
CREATE VIEW young_clerks
    AS SELECT ename, sex, deptno, phone FROM emp
        WHERE age <= 30 AND job = 'CLERKS';
```

SCOTT 不给 TOM 发放 EMP 上的访问授权，而给他发放 YOUNG_CLERKS 上的访问授权：

```
GRANT SELECT ON young_clerks TO tom
```

TOM 可以执行以下操作查询职员信息：

```
SELECT * FROM young_clerks
```

例 6.43 通过视图 YOUNG_CLERKS 对表 EMP 实现了基于内容的访问控制，它只允许用户 TOM 查询 EMP 中的 ename、sex、deptno 和 phone 字段的信息，这是 EMP 中字段信息的一个子集，而且，它只允许 TOM 查询 EMP 中年龄不超过 30 岁且岗位为 CLERKS 的记录信息，这则是 EMP 中记录信息的一个子集。视图 YOUNG_CLERKS 就是表 EMP 上的一个窗口，TOM 通过这个窗口访问到了 EMP 中的信息的一个子集。

用户 TOM 对视图 YOUNG_CLERKS 的访问，经过视图合成操作，最终转化为对用于定义该视图的基本表 EMP 的访问。例 6.43 对视图的访问最终转化成的访问是：

```
SELECT ename, sex, deptno, phone FROM emp
    WHERE age <= 30 AND job = 'CLERKS';
```

如果对视图的访问操作中带有 WHERE 条件子句，则该条件子句与定义视图的查询操作中的 WHERE 条件子句通过 AND 布尔连接进行合并，合并后的 WHERE 条件子句对访问的结果进行过滤。

在例 6.43 中，如果 TOM 执行以下操作查询职员信息：

```
SELECT * FROM young_clerks
    WHERE deptno != 20;
```

那么，合并后的 WHERE 条件子句是：

```
WHERE deptno != 20
    AND age <= 30 AND job = 'CLERKS';
```

对视图的访问最终转化成的访问是：

```
SELECT ename, sex, deptno, phone FROM emp
    WHERE deptno != 20
        AND age <= 30 AND job = 'CLERKS';
```

下面给出例 6.42 中的访问控制需求的实现方法。

例 6.44　设属主对用户 SCOTT 的职员信息表 EMP 和项目信息表 PROJECTS 的定义为

EMP：ename, job, deptno, sal, sex, age, phone, …

PROJECTS：pname, manager, ename, …

EMP 中字段的含义见例 6.43 的说明，PROJECTS 各字段表示的分别是项目名称、经理姓名、参加项目的职员姓名等。SCOTT 通过以下操作创建视图 PROJECTS_PEOPLE：

```
CREATE VIEW projects_people
    AS SELECT  e.*  FROM  emp  e, projects  p
        WHERE p.manager =
                SYS_CONTEXT ('userenv', 'session_user')
            AND e.ename = p.ename;
```

其中，SYS_CONTEXT 是 Oracle 数据库管理系统提供的系统函数，该函数的以上调用返回当前会话用户的用户名。

如果用户 ALICE 是项目经理，SCOTT 给她发放 PROJECTS_PEOPLE 上的访问授权：

```
GRANT SELECT ON projects_people TO alice
```

ALICE 可以执行以下操作查询职员信息：

```
SELECT * FROM projects_people
```

为了实现例 6.42 中给出的需求，首先要确定执行查询操作的用户是谁，因为要根据这一点来确定他所负责的项目及参加这些项目的职员，只有这些职员的信息才是该用户可以查看的信息。为此，借用 SYS_CONTEXT 系统函数，它能告诉我们当前会话中的用户是谁，而这个会话用户就是执行查询操作的用户。

SCOTT 授权 ALICE 在视图 PROJECTS_PEOPLE 上执行查询操作，ALICE 可以通过该视图来查询表 EMP 中的信息，但她所查询到的都是参加她所负责的项目的职员的信息，而其他没有参加她负责的项目的职员的信息，她是看不到的。

例 6.44 用到了当前会话中的会话用户的信息，这属于环境参数信息，所以，例 6.44 也可以看成是基于环境参数的访问控制的一个例子。

6.3.3　基于视图的写访问控制

把数据库对象上的 SELECT 操作称为读操作，而把 INSERT、UPDATE 和 DELETE 操作称为写操作。6.3.2 节介绍了基于视图的读访问控制，下面讨论基于视图的写访问控制问题。以例 6.45 中的访问控制策略为讨论的基础。

例 6.45　个人信息访问控制策略：允许用户以读和写的方式访问职员信息，但只允许他访问自己的信息，不能访问他人的信息。

可以通过视图来实现这个访问控制策略。

例 6.46　设属主对用户 SCOTT 的职员信息表 EMP 的定义为

```
EMP：ename, job, deptno, sal, sex, age, phone, …
```

SCOTT 通过以下操作创建视图 EMP_RW：

```
CREATE VIEW emp_rw
    AS SELECT * FROM emp
        WHERE ename =
            SYS_CONTEXT ('userenv', 'session_user')
                WITH CHECK OPTION;
```

这个视图过滤掉了 EMP 中除会话用户以外的其他职员的信息。其中的 CHECK 选项确保写操作能满足视图定义中的条件，使通过视图写入的信息一定能够通过视图读出来。

例 6.47　用户 SCOTT 执行以下操作，给用户 BOB、ALICE 和 TOM 发放视图 EMP_RW 上的访问授权，授权他们执行 SELECT、INSERT、UPDATE 和 DELETE 操作。

```
GRANT SELECT, INSERT, UPDATE, DELETE
    ON emp_rw TO bob, alice, tom;
```

例 6.48　用户 BOB 执行以下操作来查询职员信息：

```
SELECT ename, sal FROM EMP_RW;
```

请问：查询操作能否成功执行？

解答：查询操作可以成功执行，如果表 EMP 中含有 BOB 的信息，该查询操作将得到类似下面的结果：

```
ENAME                SAL
-------------------- -------------
BOB                  3500
```

例 6.49　用户 ALICE 执行以下操作，更新职员信息。

```
UPDATE emp_rw SET phone = '62511279';
```

请问：更新操作能否成功执行？

解答：更新操作可以成功执行，ALICE 的电话号码更新为 62511279。

例 6.50　用户 ALICE 执行以下操作，更新职员信息。

```
UPDATE emp_rw SET ename = 'BILL'
    WHERE ename = 'TONY';
```

请问：更新操作能否成功执行？

解答：更新操作不能成功执行，因为 ALICE 试图更新 TONY 的信息。

例 6.51　用户 TOM 执行以下操作，删除职员信息。

```
DELETE FROM emp_rw;
```

请问：删除操作能否成功执行？

解答：删除操作可以成功执行，TOM 的信息被删除。

例 6.52　用户 TOM 执行以下操作，删除职员信息。

```
DELETE FROM emp_rw WHERE ename = 'BILL';
```

请问：删除操作能否成功执行？

解答：删除操作不能成功执行，因为 TOM 试图删除 BILL 的信息。

例 6.53　用户 TOM 执行以下操作，插入职员信息：

```
INSERT INTO emp_rw (ename, sex, sal)
    VALUE ('TOM', 'MALE', 5000);
```

请问：插入操作能否成功执行？

解答：插入操作可以成功执行。

例 6.54　用户 TOM 执行以下操作，插入职员信息：

```
INSERT INTO emp_rw (ename, sex, sal)
    VALUE ('MARY', 'FEMALE', 5000);
```

请问：插入操作能否成功执行？

解答：插入操作不能成功执行，因为 TOM 试图插入的信息是属于 MARY 的。

可见，视图机制除了广泛用于进行 SELECT 访问控制外，也可以用于进行 INSERT、UPDATE 和 DELETE 等访问控制，这样的访问控制可以确保插入后的信息、更新后的信息、删除前的信息一定是能够通过相应视图查询得到的信息。

6.3.4　视图机制的作用和不足

关系数据库管理系统的视图机制在安全方面的主要作用，就是支持基于内容的访问控制，它能够在高层次上以一种与查询语言一致的语言来表示基于内容的访问控制策略，使低层数据与高层策略之间具有一定的独立性。在视图机制中，对数据的修改不会导致对访问控制策略进行修改，相反，如果有满足指定策略的数据插入到数据库对象中，它们就会自动地体现到有关视图的返回数据结果中。

利用关系数据库管理系统的视图机制实现访问控制的主要问题是：需要创建的视图比较多，系统的实现和维护比较复杂。这主要反映在以下几个方面：

（1）对于不同的访问类型，可能需要创建不同的视图；

（2）对于不同的视图，可能需要发放不同的授权；

（3）需要确保应用程序能够为指定的访问类型选择正确的视图。

对于不同的用户、不同的访问类型，所要实施的访问控制策略可能是不同的。例如，对于 SELECT 类型的访问，可能是允许用户查询所有记录，对于 INSERT 和 UPDATE 类型的访问，可能是允许用户改动本部门的记录，对于 DELETE 类型的访问，可能是只允许用户删除自己的记录，这样，就需要分别创建相应的 SELECT、INSERT、UPDATE 和 DELETE 视图。

视图也是数据库对象，用户需要获得相应的对象级访问授权，才能使用视图。针对不同的视图，可能需要发放不同的授权。即，针对 SELECT 视图，应该发放 SELECT 授权；针对 DELETE 视图，则应该发放 DELETE 授权。

由于存在多种类型的视图，为了使系统能够正确地工作，必须要有好的措施让应用程序选择正确的视图执行正确的操作。即，因该在 SELECT 视图上进行 SELECT 操作，在 DELETE 视图上进行 DELETE 操作，而不应该在 SELECT 视图上进行 DELETE 操作。

通过 6.3.2 节和 6.3.3 节中的例子，多少也能够体会得出来这些问题。随着用户角色和安全策略的多样化，问题的复杂性会进一步加剧。

6.4　基于角色的访问控制

在前面有关授权方面的知识的介绍中，授权是直接发放给用户的。为了使用户有权访问数据库系统中的信息，需要一一给每个用户发放合适的访问授权。可想而知，在这样的授权方式下，当用户的数量很大时，授权管理工作的任务是异常繁重的。为了减轻授权管理工作的压力，人们想到了根据岗位职责实行授权管理的方案。

6.4.1　RBAC的基本思想

根据岗位职责进行授权，就是把访问控制授权分配到工作岗位上，而不是分配给用户个人。谁在某个工作岗位上工作，谁就拥有分配到该工作岗位上的授权。当一个用户调离某个工作岗位时，他就自动失去分配到该工作岗位上的授权。

把岗位职责抽象成角色（Role）的概念，从根据岗位职责进行授权的方案中就可以抽象出基于角色的访问控制（RBAC，Role-Based Access Control）思想，进而建立基于角色的访问控制模型。

在 RBAC 模型中，访问控制授权是发放给角色的，而不是直接发放给用户的，用户通过担当某个角色来享用该角色所获得的授权。

例 6.55　设 TELLER（出纳）是某银行中的一个角色，MARY 是该银行中的一个职

员，TELLER 角色已经获得了在账目表 ACCOUNT 上的 SELECT 访问授权。请问：如何使 MARY 有权查询 ACCOUNT 中的信息？

解答：只须让 MARY 担当 TELLER 角色，她就有权查询 ACCOUNT 中的信息。

在现实世界中，角色是工作岗位的抽象，一个角色代表着一个岗位的职责及该岗位所要开展的工作。在基于角色的访问控制中，角色是访问控制授权的接受体，一个角色代表着访问控制授权的一个集合。

授权与角色之间是多对多的关系，一个授权可以发放给多个角色，一个角色可以接受多个授权。角色与用户之间也是多对多的关系，一个角色可以指派给多个用户，一个用户可以担当多个角色。

例 6.56　设 DEVPS（开发人员）和 SALES（销售人员）是某软件公司中的两个角色，把职员信息表 EMP 上的 SELECT 授权发放给 DEVPS 和 SALES 角色，把软件文档信息表 SOFTDOC 上的 UPDATE 授权发放给 DEVPS 角色，把订单信息表 ORDERS 上的 UPDATE 授权发放给 SALES 角色，请说出授权与角色的对应关系。

解答：授权到角色的对应关系是：

（1）(SELECT ON EMP) → DEVPS, SALES；

（2）(UPDATE ON SOFTDOC) → DEVPS；

（3）(UPDATE ON ORDERS) → SALES。

角色到授权的对应关系是：

（4）DEVPS → (SELECT ON EMP), (UPDATE ON SOFTDOC)；

（5）SALES → (SELECT ON EMP), (UPDATE ON ORDERS)。

在（1）中，一个授权对应两个角色，在（4）和（5）中，一个角色对应两个授权。

例 6.57　设 EMPLOYEE（职员）和 TELLER（出纳）是某银行的两个角色，用户 BOB、ALICE 和 TOM 都是该银行的职员，而 ALICE 是该银行的出纳，请说出角色与用户间的对应关系。

解答：角色到用户的对应关系是：

（1）EMPLOYEE → BOB, ALICE, TOM；

（2）TELLER → ALICE。

用户到角色的对应关系是：

（3）BOB → EMPLOYEE；

（4）ALICE → EMPLOYEE, TELLER；

（5）TOM → EMPLOYEE。

在（1）中，一个角色对应三个用户，在（4）中，一个用户对应两个角色。

虽然一个用户可以对应多个角色，但在一个系统中，尤其是在大型系统中，用户的

数量总是比角色的数量多得多，只给角色发放授权比直接给用户发放授权的工作量将少得多，因此，基于角色的访问控制能大大减少授权管理的工作量。

另外，用户所能行使的访问权限是由他所担当的角色决定的。当一个用户调离一个工作岗位时，只要切断他与所担当的角色之间的关系，就使他失去了行使该角色所赋予的权限的能力，无须特意进行撤销授权的工作。通过角色来发放授权，还能够免除因撤销某个用户的授权而引发递归回收授权的麻烦。

可见，与传统面向用户的授权方式相比，基于角色的访问控制能缓解授权管理工作的压力，减低授权管理工作的复杂性。

6.4.2　RDBMS中的RBAC

引入角色概念后，关系数据库管理系统的访问控制机制可以通过创建角色、给角色发放授权和给用户分配角色等功能来配合进行访问控制。

例 6.58　以下操作创建 TELLER 角色：

```
CREATE ROLE teller
```

以下操作给 TELLER 角色发放 ACCOUNT 表上的 SELECT、INSERT 和 UPDATE 授权：

```
GRANT SELECT, INSERT, UPDATE
    ON account TO teller
```

以下操作给用户 MARY 分配 TELLER 角色：

```
GRANT teller TO mary
```

请问：用户 MARY 拥有什么样的访问权限？

解答：因为把 TELLER 分配给 MARY 后，MARY 属于 TELLER 的成员，所以她拥有 TELLER 所获得的权限，即在 ACCOUNT 上执行 SELECT、INSERT 和 UPDATE 操作的权限。

从例 6.58 可以看出，给角色发放授权的方法与前面学过的给用户发放授权的方法是类似的，不同之处只是以角色名代替了用户名。实际上，面向用户的各种授权管理方法都可以应用到面向角色的授权管理中，只需在出现用户名的地方换上角色名即可。下面是回收角色的授权的一个例子。

例 6.59　在例 6.58 的基础上，执行以下操作回收 TELLER 角色在 ACCOUNT 上的 INSERT 授权：

```
REVOKE INSERT FROM teller
```

请问：用户 MARY 拥有什么样的访问权限？

解答：TELLER 原来拥有在 ACCOUNT 上的 SELECT、INSERT 和 UPDATE 授权，

回收了 INSERT 授权后，还剩下 SELECT 和 UPDATE 授权，所以，MARY 拥有在 ACCOUNT 上的执行 SELECT 和 UPDATE 操作的权限。

从例 6.58 中还可以看出：给用户分配角色的方法与给用户发放授权的方法是类似的，只是以角色名代替了授权名。分配角色与发放授权相对应，撤销角色与回收授权相对应。联想到角色的含义，这样的对应关系是很自然的，因为角色就是授权的集合。

例 6.60　假设在例 6.58 之前用户 MARY 没有任何访问权限，执行以下授权，撤销用户 MARY 担当的 TELLER 角色：

```
REVOKE teller FROM mary
```

请问：用户 MARY 拥有什么样的访问权限？

解答：MARY 拥有的访问权限是担当 TELLER 角色的结果，现在她不再担当该角色了，因而，她没有任何访问权限。

关系数据库管理系统通常同时支持基于角色的授权和直接给用户授权，因而，一个用户拥有的授权由以下两部分构成：

（1）直接发放给该用户的授权；

（2）分配给该用户的所有角色所拥有的授权。

角色既可以分配给用户，也可以分配给另一个角色，因而，一个角色拥有的授权由以下两部分构成：

（1）直接发放给该角色的授权；

（2）分配给该角色的所有其他角色所拥有的授权。

例 6.61　设 EMPLOYEE、TELLER 和 MANAGER 是三个角色，BOB、ALICE 和 TOM 是三个用户，ACCOUNT 是账目信息表，这些角色和用户目前没有任何授权，执行以下操作进行授权发放和角色分配：

```
GRANT SELECT ON account TO employee
GRANT UPDATE ON account TO teller
GRANT DELETE ON account TO manager
GRANT employee TO teller WITH GRANT OPTION
GRANT teller TO manager WITH GRANT OPTION
GRANT employee TO bob
GRANT teller TO alice
GRANT manager TO tom
GRANT INSERT ON account TO tom
```

请问：以上操作成功执行后，BOB、ALICE 和 TOM 分别拥有 ACCOUNT 上的什么授权？

解答：BOB 拥有 SELECT 授权，ALICE 拥有 SELECT 和 UPDATE 授权，TOM 拥

有 SELECT、INSERT、UPDATE 和 DELETE 授权。

在例 6.61 中，通过把一个角色分配给另一个角色，使角色 EMPLOYEE、TELLER 和 MANAGER 构成了一个角色链，TELLER 继承了 EMPLOYEE 中的授权，MANAGER 继承了 TELLER 中的授权，因而，MANAGER 拥有了分别发放给三个角色的所有授权。

6.4.3　角色授权与非递归式授权回收

授权可以由用户发放，也可以由角色发放。在默认情况下，授权操作是由当前会话用户执行的。在例 6.61 中，假设当前会话用户是 SCOTT，那么，例中的授权都是由用户 SCOTT 发放的，例中的角色都是由用户 SCOTT 分配的。

为了能够让某个角色来发放授权，首先必须使该角色成为当前会话角色。通常，当前会话角色是 NULL。

例 6.62　设当前会话用户是 SCOTT，SCOTT 是角色 SYSADM 的成员，执行以下操作可以使 SYSADM 成为当前会话角色：

```
SET ROLE sysadm
```

注意：只有角色中的成员才能使一个角色成为当前会话角色，例 6.62 中，如果用户 SCOTT 不是角色 SYSADM 中的成员，那么，SET ROLE 操作的执行将以失败告终。

就算当前会话角色不是 NULL，在默认情况下，授权依然由当前会话用户发放，要想由当前会话角色发放授权，还需要在授权操作中给出明确的说明。

例 6.63　设当前会话用户是 TOM，TOM 是角色 MANAGER 的成员，现执行以下操作使 MANAGER 成为当前会话角色：

```
SET ROLE manager
```

请问：执行以下两个操作的效果有什么不同？

（1）GRANT teller TO julia;

（2）GRANT teller TO julia

```
    GRANTED BY CURRENT_ROLE;
```

解答：第一个操作由用户 TOM 执行，他把角色 TELLER 分配给用户 JULIA。第二个操作由角色 MANAGER 执行，它把角色 TELLER 分配给用户 JULIA。

那么，在例 6.63 中，由用户 TOM 给 JULIA 分配角色与由角色 MANAGER 给 JULIA 分配角色有什么区别呢？区别将在撤销用户 TOM 的授权时体现出来。

如果 JULIA 中的角色 TELLER 是由用户 TOM 分配的，在撤销 TOM 中的角色 MANAGER 时，由于递归式撤销效应，也将撤销 JULIA 中的角色 TELLER。即撤销 TOM 的授权会导致撤销 JULIA 的授权。

如果 JULIA 中的角色 TELLER 是由角色 MANAGER 分配的，在撤销 TOM 中的角

色 MANAGER 时，JULIA 中的角色 TELLER 不受影响。即撤销 TOM 的授权对 JULIA 的授权没有影响。

可见，通过角色来执行授权操作能为授权的非递归式回收提供有效的支持。

6.5　数据库数据的推理控制

前面介绍了数据库系统访问控制方面的有关知识。访问控制机制能够防止用户对数据库数据进行非法直接访问，但是，没有办法防止用户对数据库数据进行非法间接访问。借助推理（Inference）的方法，用户有可能间接访问到数据库中的数据。本节要讨论的就是数据库数据间接访问的问题，即数据库数据的推理问题。

数据库中的推理问题是数据库安全中的一个棘手的问题。这里所说的推理指的是由非敏感数据推断出敏感数据的方法。推理问题的研究最初是从统计数据库开始的。随后，针对多级安全数据库中的推理问题，也有大量的研究工作。通用数据库中的推理问题，也受到了广泛的关注。在数据挖掘与 Web 应用中，同样存在着推理的问题，也需要深入研究和解决。

简单地说，统计数据库就是用于统计分析目的的数据库。统计数据库允许用户查询聚集类型的信息（如合计、平均值等），但不允许查询单个记录的信息。例如，查询员工工资的平均值是允许的，而查询某个员工的工资就是非法的了。

本节主要通过统计数据库的相关内容来认识数据库中的推理问题。为了更直观地说明问题，我们借用一张有关员工统计信息的数据库样表 EMPSTA，表所包含的字段如下：

EMPSTA：NAME, SEX, NATI, DEPT, SAL, BONUS, ABS

各字段依次表示员工的姓名、性别、民族、部门、工资、奖金和缺勤次数。假设 SAL、BONUS 和 ABS 是敏感字段，不允许查看任何一个具体员工的这些字段信息。表 6-1 是参加统计调查的员工信息的一个数据样例。

表 6-1　员工统计信息表数据样例

NAME	SEX	NATI	DEPT	SAL	BONUS	ABS
赵山	男	汉	销售部	5500	5000	1
钱河	男	苗	开发部	6500	0	0
孙娟	女	瑶	财务部	6000	3000	0
李海	男	苗	开发部	7000	1000	3
周丽	女	汉	销售部	5100	2000	1
吴梅	女	汉	财务部	6500	1000	0
郑洋	男	汉	财务部	8500	4000	3
王芳	女	苗	销售部	6500	5000	2
冯倩	女	汉	财务部	4500	0	1
陈颖	女	瑶	开发部	5000	0	2
朱峰	男	汉	开发部	6000	2000	2

6.5.1　数据库数据的推理方法

通常，在一个数据库中，求和值、记录数、平均值、中位数等，是可以发布的合法的统计数据。然而，根据这些合法的统计数据，运用推理方法，有时可以推导出不合法的敏感信息。下面依次介绍相应的推理方法。

1．借助求和结果进行推理

下面这个例子给出的是根据数据的和推断出单个记录的敏感信息。

例 6.64　以部门和性别为单位计算表 EMPSTA 中员工奖金数之和，可以得出如表 6-2 所示的合法结果，请问：根据该结果能否推断出员工陈颖的奖金数？

表 6-2　以部门和性别为单位的奖金数之和

	销　售　部	开　发　部	财　务　部	合　　计
男	5000	3000	4000	12 000
女	7000	0	4000	11 000
合计	12 000	3000	8000	23 000

解答：从奖金数的求和结果可以看出，开发部女员工的奖金数之和为 0，由此可以推断，开发部任何一个女员工的奖金数一定为 0，陈颖是开发部的女员工，因而，她的奖金数为 0。

2．借助记录个数进行推理

下面这个例子能够根据数据的和与记录的个数推断出单个员工的敏感信息。

例 6.65　以部门和性别为单位计算表 EMPSTA 中员工的记录数，可以得出如表 6-3 所示的合法结果，请问：根据该结果和表 6-2 的结果能否推断出员工赵山和郑洋的奖金数？

表 6-3　以部门和性别为单位的员工记录数

	销　售　部	开　发　部	财　务　部	合　　计
男	1	3	1	5
女	2	1	3	6
合计	3	4	4	11

解答：从员工记录数的结果可以看出，销售部和财务部都只有一个男员工，据此和表 6-2 中的奖金数统计结果，可以推断出，销售部那位男员工和财务部那位男员工的奖金数分别为 5000 元和 4000 元，由于姓名、性别和部门不是敏感信息，我们可以根据性别和部门查询姓名，查出对应的两个员工分别为赵山和郑洋，所以，可以推断出赵山和郑洋的奖金数分别为 5000 元和 4000 元。

3．借助平均值进行推理

下面这个例子给出的是根据平均值推算单个员工敏感信息的一个简单示例。

例 6.66　假设根据员工统计信息表 EMPSTA，可以报告参加统计调查的全体员工的平均工资为 6100 元，除总经理以外的其他所有参加统计调查的员工的平均工资为 5860 元，请问：根据平均工资结果能否推断出总经理的工资？

解答：表中记录总数不是敏感信息，根据表 EMPSTA 可以查询到参加统计调查的员工总数为 11，显然，总经理的工资可以由以下简单计算得到

$$6100\times11-5860\times10=8500$$

所以，可以推断出总经理的工资为 8500 元。

4．借助中位数进行推理

根据中位数推断单个记录敏感信息的一般方法，通常是寻找能得出中位数的若干查询，以求相应查询结果有唯一的交集，而交集对应的记录就是中位数所在记录。

例 6.67　针对表 EMPSTA 的样例数据，请问：以下查询结果中奖金的中位数是什么？

```
SELECT NAME, SEX, ABS, BONUS
    FROM empsta
        WHERE SEX = '男'
            ORDER BY BONUS;
```

解答：查询结果如表 6-4（a）所示，奖金中位数是 2000 元。

表 6-4　含中位数的两个列表

NAME	SEX	ABS	BONUS
钱河	男	0	0
李海	男	3	1000
朱峰	男	2	2000
郑洋	男	3	4000
赵山	男	1	5000

（a）

NAME	SEX	ABS	BONUS
陈颖	女	2	0
朱峰	男	2	2000
王芳	女	2	5000

（b）

例 6.68　针对表 EMPSTA 的样例数据，请问：以下查询结果中奖金的中位数是什么？

```
SELECT NAME, SEX, ABS, BONUS
    FROM empsta
        WHERE ABS = 2
            ORDER BY BONUS;
```

解答：查询结果如表 6-4（b）所示，奖金中位数是 2000 元。

当然，例 6.67 和例 6.68 是为了说明问题而设计的，由于 ABS 和 BONUS 都是敏感

字段，所以，一般情况下，用户是不能执行那两个查询的。

在表 6-4 中，查询结果（a）与查询结果（b）有唯一的交集，那就是员工朱峰对应的记录信息，该记录信息中的奖金值在两个查询结果中都是中位数。可见例 6.67 和例 6.68 分别给出的两个查询是能得出中位数的查询，且两个查询结果中的中位数对应同一个记录，该记录信息就是两个查询结果的唯一交集。

如果知道有中位数的若干查询结果有唯一的交集，就有可能从中推断出敏感信息。

例 6.69　假设以下两个是允许的查询中位数的操作：

```
SELECT median(BONUS)
    FROM empsta WHERE SEX = '男';
SELECT median(BONUS)
    FROM empsta WHERE ABS = 2;
```

两个查询结果都是 2000，如果知道男员工与缺勤次数为 2 的员工有唯一的交集，且交集就是奖金中位数记录，又知朱峰的缺勤次数为 2，请问：能否推断出哪位员工的奖金是 2000 元？

解答：因为姓名和性别不是敏感信息，用户可以执行以下操作：

```
SELECT NAME FROM empsta WHERE SEX ＝ '男'
```

查询结果显示朱峰是男员工，又知朱峰的缺勤次数为 2，因此，朱峰是男员工与缺勤次数为 2 的员工的唯一交集，所以，朱峰的奖金数是中位数，而已知查询结果显示奖金中位数是 2000，故此，朱峰的奖金是 2000 元。

5．借助智能填充进行推理

在数据库查询中，有时，满足条件的记录并不多，这样，数量很少的记录就在查询结果中占了很高的比例。在极端情况下，只有 1 条记录满足条件，1 条记录就是查询结果的 100%。作为防止敏感信息泄露的措施之一，数据库管理系统可以屏蔽数量少但比例高的数据。

为了防止数据库管理系统屏蔽掉数量少但比例高的查询结果，攻击者可能采取智能填充的办法愚弄数据库管理系统。也就是说，当攻击者需要得到的是某个特定的值时，他并不直接查询该值，假设数据库中共有 n 个值，他会去查询其他的（n–1）个值，然后根据整体的 n 个值和查询的（n–1）个值计算出该特定的值。

例 6.70　在一个限制“数量少但比例高”的信息的数据库管理系统中，设用户执行以下操作查询员工数信息：

```
SELECT count(*) FROM empsta
    WHERE SEX = '女'
        AND NATI = '汉'
        AND DEPT = '销售部';
```

请问：数据库管理系统是否提供查询结果？

解答：数据库管理系统查询后发现，只有 1 条记录满足条件，即 1 条记录构成了100%的查询结果，符合“数量少但比例高”的限定，所以，数据库管理系统不提供查询结果。

例 6.71　在一个限制“数量少但比例高”的信息的数据库管理系统中，设用户执行以下两个操作查询员工数信息：

```
SELECT count(*) FROM empsta
    WHERE SEX = '女';
SELECT count(*) FROM empsta
    WHERE SEX = '女'
        AND (NATI != '汉'
                OR DEPT != '销售部');
```

请问：数据库管理系统是否提供查询结果？

解答：数据库管理系统查询后发现，两个查询都有多条记录满足条件，没有出现“数量少但比例高”的情况，所以，数据库管理系统提供查询结果，第一个结果是 6，第二个结果是 5。

例 6.70 中的查询结果被隐藏了，但例 6.71 的查询结果能够正常获得。那么，这两个例子有什么关系呢？

实际上，例 6.70 的查询条件的形式是：

$$a \wedge b \wedge c$$

根据逻辑运算规则，有

$$\begin{aligned} & a \wedge b \wedge c \\ & = a - (a \wedge \neg\ (b \wedge c) \\ & = a - (a \wedge (\neg\ b \vee \neg\ c)) \end{aligned}$$

例 6.71 的第一个查询条件是的形式是：

$$a$$

第二个查询条件的形式是：

$$(a \wedge (\neg\ b \vee \neg\ c))$$

也就是说，例 6.71 中的第一个查询的结果减去第二个查询的结果等价于例 6.70 的查询结果。虽然例 6.70 的查询结果被隐藏了，但是，例 6.71 的查询结果可以正常获得，所以，可以由例 6.71 的查询结果推断出例 6.70 所要的查询结果，即

$$6 - 5 = 1$$

从而得知，例 6.70 的查询实际上得到的员工数应该是 1。

6．借助线性特性进行推理

数据库管理系统能比较容易地实现敏感信息的直接访问控制，可以禁止一般用户直接查询数据库中的敏感信息。敏感信息经过算术运算后，可以降为非敏感信息。例如，a_1 和 a_2 都是敏感信息，但（a_1+a_2）可能就不是了。但是，如果允许查询敏感信息经过算术运算后的结果，从这些表面看来不是敏感信息的结果中，也许可以推出敏感信息来。

例 6.72　设 c_1、c_2、c_3、c_4 和 c_5 都是表 TBL 中的敏感字段，但以下每个查询都是允许的合法查询（其中 P 是一个谓词）：

```
SELECT c1+c2+c3+c4+c5 FROM tbl WHERE P;
SELECT c1+c2+c4 FROM tbl WHERE P;
SELECT c3+c4 FROM tbl WHERE P;
SELECT c4+c5 FROM tbl WHERE P;
SELECT c2+c5 FROM tbl WHERE P;
```

如果以上的每个查询得到的结果都是一个确定的值，而且，这些值是一致的，请问：这些查询是否会泄露敏感信息？

解答：设以上查询得到的结果分别为 q_1、q_2、q_3、q_4 和 q_5，则有

$$q_1 = c_1 + c_2 + c_3 + c_4 + c_5 \quad ①$$

$$q_2 = c_1 + c_2 + c_4 \quad ②$$

$$q_3 = c_3 + c_4 \quad ③$$

$$q_4 = c_4 + c_5 \quad ④$$

$$q_5 = c_2 + c_5 \quad ⑤$$

把 c_1、c_2、c_3、c_4 和 c_5 看成未知数，这是一个普通的线性方程组，很容易求解。

①–②有

$$q_1 - q_2 = c_3 + c_5 \quad ⑥$$

③–④有

$$q_3 - q_4 = c_3 - c_5 \quad ⑦$$

⑥+⑦有　　$q_1 - q_2 + q_3 - q_4 = 2c_3$

即　　$c_3 = (q_1 - q_2 + q_3 - q_4)/2$

将 c_3 代入③中，即可求出 c_4；将 c_4 代入④中，即可求出 c_5；将 c_5 代入⑤中，即可求出 c_2；将 c_2 和 c_4 代入②中，即可求出 c_1。至此，我们得出了 c_1、c_2、c_3、c_4 和 c_5 的值。

所以，题中所列查询的组合泄露了 c_1、c_2、c_3、c_4 和 c_5 等敏感信息。

这个例子告诉我们，根据敏感信息的算术运算结果，有可能推导出敏感信息。值得注意的是：与此类似，根据敏感信息的逻辑运算结果，也有可能推导出敏感信息。

6.5.2　数据库数据的推理控制

数据库数据的推理控制就是阻止用户根据非敏感信息推导出敏感信息。在了解数据库数据的推理方法的基础上，要设法消除这些推理方法成功实施的条件，从而破坏推理企图的成功实现。

前面着重介绍了统计数据库相关的推理问题。统计推理的控制方法从本质上可以分为两大类：一类是对查询进行控制；另一类是对数据库中的数据项进行控制。对查询进行控制是一件比较困难的事情，很难确定一个给定的查询是否会泄露敏感信息。下面重点考虑对数据库中的数据项进行控制的方法。

对数据项进行控制，可以采用滤除法（Suppression）和隐藏法（Concealing）。滤除法，就是当查询可能涉及敏感信息时，默默拒绝提供查询结果，不给任何反馈。隐藏法，就是当查询可能涉及敏感信息时，隐藏掉部分与查询相关的信息，只提供近似查询结果。

滤除法要么不提供查询结果，要么提供正确的查询结果。隐藏法总是提供查询结果，但所提供的可能是不全面的结果。在面向数据项的推理控制中，选择滤除法还是隐藏法，要视数据库的环境因素而定。

1．滤除法及遇到的问题

什么样的查询结果需要滤除呢？前面曾经提到过，数量少而比例高的结果容易泄露敏感信息。例如，根据表 6-3 的显示，销售部和财务部男员工的记录数都是 1，属于典型的数量少而比例高的结果，例 6.65 告诉我们，这样的结果会泄露敏感信息。所以，滤除法应该滤除数量少而比例高的结果。

取员工统计信息表 EMPSTA 为样本，以部门和性别为单位计算员工的记录数，所得结果如表 6-3 所示，其中值为 1 的结果可能泄露敏感信息，可以滤除，滤除后的结果如表 6-5 所示。

表 6-5　经过滤除处理的员工记录数

	销　售　部	开　发　部	财　务　部	合　　计
男	—	3	—	5
女	2	—	3	6
合计	3	4	4	11

例 6.73　分析如表 6-5 所示的已经进行过滤除处理的查询结果，请问：该结果能否有效抵御数据推理？

解答：不妨用 e_{ij} 表示表 6-5 中数值部分 i 行 j 列的值。由于表 6-5 的查询结果中含有行合计和列合计，所以，根据查询结果中的数据很容易计算出滤除掉的数据，具体地说，虽然 e_{11}、e_{13} 和 e_{22} 已被滤除掉，但经过以下的简单计算就能把它们确定出来

$$e_{11} = e_{31} - e_{21} = 3 - 2 = 1$$
$$e_{13} = e_{33} - e_{23} = 4 - 3 = 1$$
$$e_{22} = e_{32} - e_{12} = 4 - 3 = 1$$

或者

$$e_{22} = e_{24} - e_{21} - e_{23} = 6 - 2 - 3 = 1$$

可见，根据现存结果可以推出滤除掉的结果，在此基础上，可以进一步推导出敏感信息。所以，表 6-5 的查询结果不能有效地抵御数据推理。

针对例 6.73 所揭示的问题，为了防止由现存结果推导出滤除掉的结果，如果查询结果中提供列合计，滤除一个单元格中的数值时，至少需要同时滤除同列上另一个单元格中的数值；如果查询结果中提供行合计，滤除一个单元格中的数值时，至少需要同时滤除同行上另一个单元格中的数值。根据这个逻辑，除了合计栏中的值以外，表 6-5 中的其他所有值都需要滤除掉。原因是这个样本结果太小，除了合计栏外，总共只有两行值。

2．结果合并式隐藏法

通过合并查询结果，可以隐藏部分数据信息，有助于防止攻击者从查询结果中推导出敏感信息。

例 6.74　基于员工统计信息表 EMPSTA 样例，按照性别和缺勤次数对员工数进行统计，得到统计结果如表 6-6（a）所示。显然，根据该结果，可以断定，任何女员工的缺勤次数都不会是 3。可以把缺勤次数为 0 次和 1 次的结果进行合并，2 次和 3 次的结果进行合并，得到如表 6-6（b）所示的结果，根据该结果，不可能推断某确定缺勤次数的员工信息。

表 6-6　员工数统计—统计结果的合并

	缺勤 0 次	缺勤 1 次	缺勤 2 次	缺勤 3 次
男	1	1	1	2
女	2	2	2	0

（a）

	缺勤 0 或 1 次	缺勤 2 或 3 次
男	2	3
女	4	2

（b）

与例 6.74 略有不同的另一种结果合并方法是按照数值的取值范围组织统计结果。以对奖金数的统计为例，可以设 0～1999 元、2000～3999 元、4000 元以上等几个取值范围，以这些取值范围为单位提供统计查询结果。根据这样的结果，不可能推出某个记录的确切数值，即使某个数值只有一个记录与之对应。

3．随机抽样式隐藏法

随机抽样式隐藏法通过随机抽样的办法隐藏数据库信息。当数据库管理系统接收到一个查询请求时，它首先从查询对象数据中生成一个随机样本数据集，然后在样本数据集上实施查询操作，生成查询结果。

随机样本数据集是查询对象的原始数据集的一个子集，查询结果是根据样本数据集生成的，而不是根据原始数据集生成的，所以，查询结果实际上只能反映样本数据集的情况，而无法完全反映原始数据集的真实情况。

例 6.75　设 TBL 是一个实现随机抽样式隐藏法的数据库系统中的一张表，某用户请求查询 TBL 中拥有性质 A 的记录数占总记录数的百分比，系统返回结果是 5%，请问：表 TBL 中拥有性质 A 的记录数与总记录数的比例是否精确等于 5%，为什么？

解答：数据库管理系统接收到用户的查询请求后，首先生成 TBL 的一个随机样本表 TBL_{sample}，然后在 TBL_{sample} 上进行查询，5%的结果是这样计算出来的

$$\frac{\text{表}TBL_{sample}\text{中拥有性质A的记录数}}{\text{表}TBL_{sample}\text{中的总记录数}} = 5\%$$

所以，表 TBL 中拥有性质 A 的记录数与总记录数的比例并不精确等于 5%。

数据库管理系统可以生成足够大的样本数据集，以确保查询的有效性。但是，无论如何，原始数据集中的部分数据已经隐藏了起来。用户获得的只是由样本数据集生成的查询结果，要想根据这样的查询结果准确推断出原始数据集中的敏感信息，困难是很大的。

直觉告诉我们，样本数据集越大，样本个数越多，就越有可能反映原始数据集的真实情况。样本数据集的大小通常是由数据库管理系统确定的，用户选择的余地有限。攻击者极有可能想方设法获取访问多个样本数据集的机会，以得到更真实的信息。

例如，就例 6.75 中的问题而言，一个样本数据集对应一个结果，如果能够访问大量的样本数据集，得到大量的结果，就可以通过求这些结果的平均值来获得更真实的结果。

如果数据库管理系统处理每个查询请求时都随机地生成样本数据集，那么，只要反复多次发出同一个查询请求，就可轻而易举地达到访问大量的样本数据集的目的，然后借助求平均值的方法推断数据库系统的真实特性。

为了防止攻击者进行此类平均值计算式的攻击，可以采取为等价查询选择相同的样本数据集的应对方法。在这种应对方法的控制下，等价的查询访问到的都是相同的样本数据集，得到的都是相同的结果，这就挫败了攻击者访问大量样本数据集的企图，达到了防止攻击者借助反复多次查询来推断数据库系统真实特征的目的。

4．偏差导入式隐藏法

数据扰动（Perturbation）是保护数据库数据安全的重要措施之一。简单地说，数据扰动就是对原始数据进行适当的修改，以便掩饰数据的真实值。数据扰动的常用技术有数据交换（Swapping）和数据随机化（Randomization）等。数据交换就是交换数据记录之间的数值。数据随机化则是给数据添加噪声。

给数据添加噪声也就是给数据导入偏差值。在根据数据库数据生成统计结果前，给数据库数据导入适当的偏差值，可以有效地隐藏数据库的真实信息。

例 6.76　设 TBL 是一个实现偏差导入式隐藏法的数据库系统中的一张表，表中第 i 个数据项的值是 x_i，偏差导入规则是为 x_i 生成一个比较小的随机偏差值 ε_i（可以大于 0，也可以小于 0），生成统计结果时，用$(x_i+\varepsilon_i)$取代 x_i，现有用户要求查询第 1～n 个数据项之和，请比较查询结果与实际结果的差异。

分析：查询所得结果是

$$\mathrm{SUM_q} = \sum_{i=1}^{n}(x_i + \varepsilon_i)$$

实际结果是

$$\mathrm{SUM_r} = \sum_{i=1}^{n}(x_i)$$

所以，查询结果 $\mathrm{SUM_q}$ 是一个与实际结果 $\mathrm{SUM_r}$ 比较接近的值，但不是等于实际结果 $\mathrm{SUM_r}$ 的精确值。

为数据库对象的数据项导入比较小的偏差值之后，查询得到的统计结果与真实的统计结果是比较接近的，但并不相等，而且，偏差值具有一定的随机性，这能较好地隐藏数据库数据的真实特性。

与随机抽样式隐藏法类似，为了防止基于反复多次查询的攻击，可以为等价查询提供相同的偏差值集合。因为保存偏差值集合比保存样本数据集容易，所以，偏差导入式隐藏法比随机抽样式隐藏法易于使用。

5．基于查询分析的推理控制

基于查询分析的推理控制，就是对查询及其潜在意义进行分析，判断是否存在推理问题，从而确定是否提供查询结果。

基于查询分析的推理控制是比较困难的，方法之一是维护每个用户的查询历史信息，处理查询请求时，要分析历史的查询结果，结合本次查询请求的特点，判断是否存在推理的可能性，如果存在推理的可能性，就拒绝实施查询请求，只有在能够确定没有推理的可能性时，才彻底实施查询操作，并最终提供查询结果。

6.6　本章小结

本章首先介绍数据库系统安全性的整体概况，然后以关系数据库系统为代表，着重从关系数据库自主访问授权、基于视图的访问控制、基于角色的访问控制和数据库数据的推理控制等方面介绍数据库系统安全性的基本思想、技术和方法。

数据库系统安全是信息系统安全的重要组成部分，既具有信息安全的共同特性，也

具有自身的特点。机密性、完整性和可用性是信息安全的经典三要素，数据库系统安全的完整解决方案必须符合机密性、完整性和可用性的需求。在数据库系统安全领域，除了经典三要素外，私密性需求也日显突出。

数据库数据的安全性由数据库管理系统的多种组件来实现支持。数据的机密性主要由访问控制机制实现，数据的完整性主要由访问控制机制和语义完整性约束来实现，数据的可用性主要由数据恢复子系统和并发控制机制来实现。抵御拒绝服务攻击的技术也能在保护数据的可用性方面发挥积极的作用。

访问控制机制是实现数据库系统安全性的重要机制。数据库系统的访问控制机制要支持基于内容的访问控制。数据库系统的访问控制可分为自主访问控制和强制访问控制。多级安全数据库系统是实现强制访问控制的典型系统。多级安全数据库系统需要考虑隐蔽信道的处理问题。

关系数据库管理系统的安全模型具有最基础和最广泛的意义。面向对象数据库管理系统的安全模型和对象关系数据库管理系统的安全模型也受到了人们的重视。

基于 XML 的 Web 数据和多媒体数据等新的数据形态给数据库系统安全提出了新的要求，新的数据应用形态也给数据库系统安全带来了新的挑战。推理控制与统计数据库安全、面向数据库外包的加密数据查询处理、支持隐私保护的数据库系统、GIS 数据的安全性、信息网格体系结构的安全性、传感数据的安全性、Web 服务和语义 Web 的安全与隐私保护等，都是数据库系统安全领域需要研究和解决的问题。

关系数据库管理系统的自主访问控制策略，根据主体的标识和授权的规则控制主体对客体的访问。关系数据库管理系统自主访问控制的重要内容是授权管理，授权管理指的是系统发放授权和回收授权的功能，该功能负责在访问控制机制中建立授权和撤销授权。

授权管理策略有集中式授权管理和基于属主的授权管理。在集中式授权管理中，只有一些拥有特权的主体能够发放和回收授权。在基于属主的授权管理中，主体对客体的访问授权的发放和回收由相应客体的属主来负责。

通常，基于属主的授权管理提供对委托授权的支持。委托授权就是一个主体把对客体的授权的发放和回收权传递给另一个主体，使另一个主体能够发放和回收对相应客体的授权。授权的委托管理涉及授权的递归回收问题。实际应用要求关系数据库管理系统提供递归式授权回收和非递归式授权回收支持。

一般情况下，数据库的访问授权指的是肯定式授权，肯定式授权在实现禁止访问的需求方面存在一定的不足，引入否定式授权可以弥补这一不足。在同时支持肯定式授权和否定式授权的系统中，需要解决授权冲突问题。

在普通的授权管理中，主体一旦获得某个授权，该授权就永远有效，直到被撤销。考虑时效因素的授权明确指定授权的时效性，限定授权只能在具体的时间段内有效，在该时间段以外，授权无效。

除了数据库对象级的访问授权以外，关系数据库管理系统还提供系统级的访问授权。系统级的访问授权实现对数据库模式的访问控制。数据库系统文件的安全性由操作系统的安全机制实现。

关系数据库管理系统中的视图机制可用于提供基于内容的访问控制支持，基于内容的访问控制要求根据数据的内容进行访问控制判定。视图机制可广泛用于进行查询访问控制，也可用于进行插入、更新和删除等访问控制。利用视图机制实现访问控制的主要问题是：需要创建的视图比较多，系统的实现和维护比较复杂。

与传统面向用户的授权方式相比，基于角色的访问控制能缓解授权管理工作的压力，降低授权管理工作的复杂性。在基于角色的访问控制中，角色是访问控制授权的接受体，一个角色代表着访问控制授权的一个集合。通过角色来执行授权操作，能为授权的非递归式回收提供有效的支持。

访问控制机制可以防止用户非法对数据库数据进行直接访问，但无法防止用户非法对数据库数据进行间接访问。推理是数据库数据面临的严重的间接访问威胁。推理指根据合法的非敏感数据推导出非法的敏感数据。

统计数据库、多级安全数据库和通用数据库中都存在推理问题，数据挖掘和 Web 应用中也存在着推理问题。

求和值、记录数、平均值、中位数等是数据库系统中可以发布的常见合法统计数据，有时，可以运用推理方法根据这些合法的统计数据推导出不合法的敏感信息。借助智能填充和线性运算有可能加大推导出敏感信息的可能性。

数据库数据的推理控制就是阻止用户根据非敏感信息推导出敏感信息。推理的控制方法从本质上可以分为两大类：一类是对查询进行控制；另一类是对数据库中的数据项进行控制。

对查询进行控制就是对查询及其潜在意义进行分析，判断是否存在推理问题，从而确定是否提供查询结果。方法之一是维护查询历史信息，根据历史查询结果和当前查询请求判断是否存在推理的可能性。

对数据项进行控制，可以采用滤除法和隐藏法。滤除法，就是当查询可能涉及敏感信息时，默默拒绝提供查询结果，不给出任何反馈。隐藏法，就是当查询可能涉及敏感信息时，隐藏掉部分与查询相关的信息，只提供近似查询结果。结果合并、随机抽样、偏差导入等是隐藏法中常见的手段。



习题 6

1. 数据库数据的机密性、完整性和可用性分别由数据库管理系统中的哪些组件来提供支持？

2. 关系数据库系统的访问控制模型与操作系统的访问控制模型主要有哪些不同？

3. 关系数据库系统的自主访问控制模型具有哪些主要特性？

4. 强制访问控制和多级安全数据库系统常以_____为基础对数据库数据进行访问控制，主要是实现_____（关系，属性，记录）级的访问控制。

5. 什么是隐蔽信道？系统中的哪些成分容易被利用来制造隐蔽信道？为什么并发控制机制有可能导致隐蔽信道？多实例机制是如何防止隐蔽信道的？

6. 结合 SQL，简要说明关系数据库系统自主访问控制中授权的发放与回收的基本方法。

7. 什么是否定式授权？什么是授权冲突？简要说明解决授权冲突的基本原则。

8. 什么是授权的委托管理？简要说明授权的递归回收和非递归回收的基本思想。

9. 简要说明具有时效性的访问控制授权的基本思想。

10. 简要说明数据库系统中系统级的访问授权主要包括哪些方面的内容。

11. 举例说明什么是基于内容的访问控制，简要说明通过视图机制实现基于内容的访问控制的基本方法。

12. 简要说明基于角色的访问控制模型的基本思想，并结合 SQL，简要说明关系数据库管理系统为该模型提供的基本支持方法。

13. 简要说明利用基于角色的访问控制机制实现授权的非递归回收的基本方法。

14. 对数据库数据进行推理攻击的常用方法主要有哪些？

15. 通过对数据库中的数据项进行控制来防止数据库推理攻击的常见方法有滤除法和隐藏法，滤除法与隐藏法的主要区别是什么？

16. 简要说明防止数据库推理攻击的以下隐藏法的基本思想：结果合并式隐藏法、随机抽样式隐藏法、偏差导入式隐藏法。

第 7 章
数据库系统的增强安全性

通过第 6 章的学习，建立了数据库系统安全的整体概念，并且对数据库系统的若干典型的基础安全技术有了比较深入的了解。在这些基础上，本章以 Oracle 实现的虚拟专用数据库机制（VPD 机制）和基于标签的安全机制（OLS 机制）为代表，介绍增强数据库系统安全性的技术和方法。VPD 机制提供对基于内容的访问控制和环境敏感的访问控制的支持，OLS 机制为多级安全数据库系统提供了一个实用的解决方案。

7.1　虚拟专用数据库机制

虚拟专用数据库（VPD，Virtual Private Database）机制是一种细粒度的、基于内容的记录级访问控制机制。它的基本思想是动态地、透明地给数据库访问语句附加上合适的谓词，以便根据实际的安全需求限定用户所能访问到的数据记录，从而达到为特定的安全需求提供特定的数据库数据记录集的效果。

由于谓词是透明地附加到数据库访问语句上去的，用户也许根本意识不到这一点，在他们看来，好像系统为他们提供了一个量身定制的专用数据库一样，如图 7.1 所示。

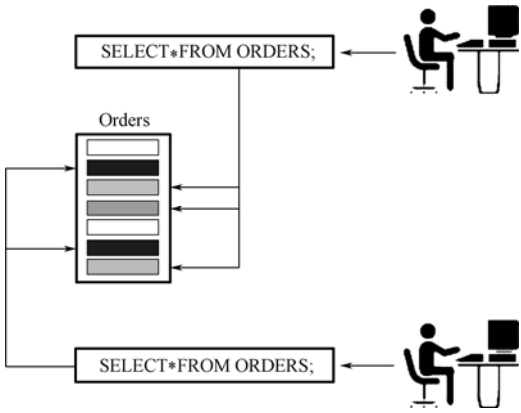


图 7.1　虚拟专用数据库思想

对数据库数据记录的访问控制方法由安全策略来确定。在 Oracle 的 VPD 机制中，一个安全策略的规则由一个函数来实现，每个函数的返回值都是一个字符串，这样的一个字符串就构成了一个谓词。一个函数定义了之后，就可以与一张表关联起来，这样，每当要对这张表进行访问时，相应的谓词就自动附加到对应的访问语句上，从而在访问控制中发挥作用。

7.1.1　初识VPD机制

数据库记录的访问控制策略总是针对数据库表定义的，对于一张企业员工信息表 SCOTT.EMP 而言，一个简单的安全策略例子是：不允许查询部门号为 10 的记录信息。下面的示例为这样的安全策略定义一个策略函数。

例 7.1　用户 SEC_MGR 执行以下操作，定义一个策略函数 no_dept10：

```
CREATE OR REPLACE FUNCTION no_dept10 (
    p_schema        IN   VARCHAR2,
    p_object        IN   VARCHAR2)
    RETURN VARCHAR2
  AS
BEGIN
  RETURN 'deptno != 10';
END;
```

如例 7.1 所示，每个策略函数都有两个参数：第一个参数是数据库模式的名称；第二个参数是数据库对象的名称，数据库对象可以是表或视图等。策略函数将与参数中给定的模式中的对象相关联。数据库系统在启动策略函数时，为策略函数的这两个参数提供参数值。

下面的示例在系统中登记一个安全策略，为例 7.1 中定义的策略函数指定参数值，从而把该策略函数与一个指定模式中的一张指定的表关联起来。

例 7.2　用户 SEC_MGR 执行以下操作，登记一个安全策略，把函数 no_dept10 与模式 SCOTT 中的表 EMP 关联起来：

```
BEGIN
  DBMS_RLS.add_policy
              (object_schema      => 'SCOTT',
               object_name        => 'EMP',
               policy_name        => 'quickstart',
               policy_function    => 'no_dept10');
END;
```

此后，对 SCOTT.EMP 进行访问时，函数 no_dept10 产生的谓词自动附加到访问语句上。下面的示例能表明这样的效果。

例 7.3　用户 SCOTT 执行以下操作，查询 EMP 表中的部门号：

```
SELECT DISTINCT deptno FROM emp;
```

查询结果是：

```
DEPTNO
----------
    20
    30
```

例 7.3 的显示结果中没有部门号 10，这是因为系统自动给其中的 SELECT 语句附加

上了函数 no_dept10 产生的谓词，实际上，把该 SELECT 语句修改成了以下形式：

```
SELECT DISTINCT deptno FROM emp WHERE deptno != 10
```

所以，执行例 7.3 中的 SELECT 语句时，部门号为 10 的记录被过滤掉了。当然，系统对 SELECT 语句的修改对于用户来说是透明的。从这个示例中还看到，这里所说的谓词就是 WHERE 子句。

策略函数可以根据需要进行修改，不需要重新登记安全策略，函数修改后即可生效。下面的示例修改例 7.1 定义的策略函数。

例 7.4　用户 SEC_MGR 执行以下操作，修改函数 no_dept10 的定义：

```
CREATE OR REPLACE FUNCTION no_dept10 (
    p_schema        IN   VARCHAR2,
    p_object        IN   VARCHAR2)
    RETURN VARCHAR2
  AS
BEGIN
  RETURN 'USER != "SYSTEM"';
END;
```

下面的示例表明例 7.4 对函数 no_dept10 的修改生效了。

例 7.5　用户 SCOTT 执行以下操作，查询表 EMP 中的记录数：

```
SELECT COUNT(*) Total_Records FROM emp;
```

查询结果是：

```
TOTAL_RECORDS
-------------------------
          14
```

用户 SYSTEM 执行以下操作，查询表 SCOTT.EMP 中的记录数：

```
SELECT COUNT(*) Total_Records FROM scott.emp;
```

查询结果是：

```
TOTAL_RECORDS
-------------------------
          0
```

由例 7.5 中的第一个查询结果可知，表 EMP 中是有记录的，可是第二个查询结果显示表中的记录数为 0，原因是第二个查询是由用户 SYSTEM 执行的，而例 7.4 修改后的函数 no_dept10 产生的谓词把用户 SYSTEM 执行的查询的结果都过滤掉了。

7.1.2　VPD机制的工作原理

VPD 机制是绑定到表、视图或同义词（Synonym）等数据库对象之上的访问控制机制，为了叙述简单，下面仅以表对象为代表进行讨论。

VPD 机制根据安全策略实施访问控制，针对需要保护的数据库表，可以为它定义多种安全策略。每个安全策略由一个策略函数来实现。每个策略函数具有两个参数，第一个参数是模式名，第二个参数是表名。通过这两个参数，一个策略函数可以与一个特定模式中的一张特定的表关联起来。每个策略函数产生一个字符串类型的返回值，这个返回值构成一个谓词。

当用户对受保护的表进行访问时，系统启动与该表关联的策略函数，策略函数产生相应的谓词，系统用该谓词构造一个 WHERE 子句，把它附加到原始的访问语句上，把原始的访问语句修改为新的访问语句。

例 7.6　设用户通过以下语句进行查询：

```
select * from EMP
```

设与表 EMP 关联的策略函数返回的谓词如下：

```
ename = USER
```

系统修改形成的、实际执行的语句是：

```
select * from EMP where ename = USER
```

策略函数的参数值是在对安全策略进行登记时确定的，安全策略的登记通过存储过程 DBMS_RLS.ADD_POLICY 来完成。在登记安全策略时，需要给出模式名、表名、安全策略名、策略函数名等信息，如例 7.2 所示。

1．基于访问类型的访问控制

安全策略可以明确地针对 SELECT、INSERT、UPDATE、DELETE 或 INDEX 等具体类型的语句来实施。其中，针对 INDEX 类型语句的安全策略将影响 CREATE INDEX 和 ALTER INDEX 等 DDL 命令的操作。

当用户对受保护的表进行访问时，VPD 的访问控制机制就默默地开始工作，检查访问语句的类型和相关安全策略，执行对应的策略函数，修改原始访问语句，并执行修改后的访问语句。图 7.2 是 VPD 机制的访问控制过程的一个示例。

在默认情况下，一个安全策略将实施到所有类型的数据库访问语句中，对所有类型的访问语句都实施控制。例如，例 7.2 中登记的安全策略对所有类型的访问语句都实施控制。

如果要使一个安全策略只针对 SELECT、INSERT、UPDATE、DELETE 或 INDEX 中的一类或几类语句实施控制，可以在登记安全策略时明确说明。如果在登记一个安全

策略时，通过 STATEMENT_TYPES 指定了访问语句的类型，那么，该安全策略将只对指定类型的语句起作用。这样，可以实现基于访问类型的安全策略。

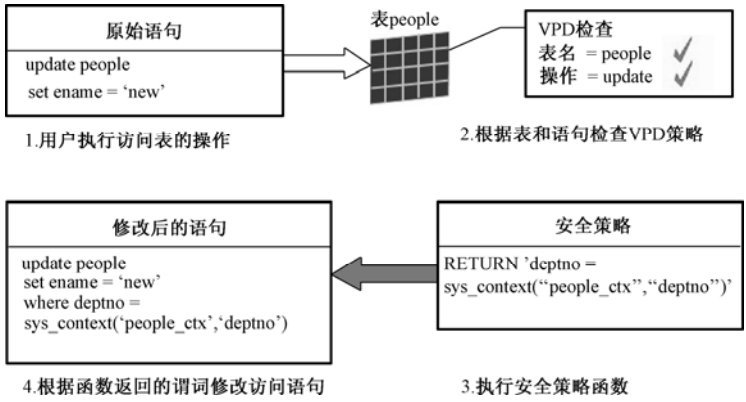


图 7.2　VPD 机制的访问控制过程

例如，可以制订一个安全策略，允许 SELECT 语句访问所有的记录；另外，制订一个 INSERT 和 UPDATE 型安全策略，使用户只能对他所在部门的记录进行插入或更新操作；并且，制订一个 DELETE 型安全策略，使用户只能删除他自己的记录。

2．多个安全策略的访问控制

可以对同一张表实施多个安全策略。当多个安全策略实施到同一张表上时，系统对多个安全策略的控制实行逻辑“与”操作。如下面的示例所示。

例 7.7　一个安全策略 A 获得的返回值是：

ename = USER

另一个安全策略 B 获得的返回值是：

sal > 2000

如果 A 和 B 这两个安全策略实施在同一张表的同一类操作上，那么，系统最终生成的谓词将是：

where ename = USER and sal > 2000

有时两个安全策略可能会得到冲突的结果。如下面的示例所示。

例 7.8　一个安全策略 A 获得的返回值是：

deptno = 20

另一个安全策略 B 获得的返回值是：

deptno = 10

如果 A 和 B 这两个安全策略实施在同一张表的同一类操作上，那么，系统最终生成的谓词将是：

where deptno = 20 and deptno = 10

因为 deptno 的值不可能既等于 20 又等于 10，所以，这个 WHERE 子句中的两个条件是冲突的，在这种情况下，WHERE 条件为假，相应的数据库访问语句将访问不到表中的任何记录。

为了处理安全策略可能存在的结果冲突问题，可以在 VPD 机制中对安全策略实行分组管理。分组时，确保同一个组中的安全策略不产生冲突的结果。根据具体的应用环境，启用一些安全策略和禁用一些安全策略，可以避免安全策略冲突情况的发生。可以通过在环境参数中配置安全策略的方法来达到这一目的。

每个会话都有相应的环境参数，把安全策略的名称设置到会话的环境参数中，可以确定在一个会话中要实施的将是哪些安全策略。在建立一个会话时，系统自动启用该会话的环境参数中设置的那些安全策略，从而提供基于会话的安全策略支持，解决安全策略冲突的问题。

3．VPD机制的安全机理

VPD 机制的安全性来自谓词的记录过滤作用，不管数据库的访问请求是怎样发出的，也不管访问请求是哪个用户发出的，谓词都要对原始的查询结果进行过滤。这种记录过滤机制实现了独立于应用的、一致的记录级安全性，即不管与数据库数据打交道的是什么样的应用，VPD 机制都要实施一致的记录过滤控制。VPD 机制的安全性与受保护的数据紧密地结合在一起，它实施的记录过滤过程对发出数据库访问请求的应用是透明的，这种安全性具有不可旁路性。

记录过滤并不是 VPD 机制独有的特性，它是数据库系统中的一个基本功能，可以用多种方法来实现。在应用程序中实现记录过滤是一种常见的方法。与基于应用的记录过滤方法相比，VPD 机制提供的是一种紧贴数据库的低层控制方法，更有利于实现一致和持久的安全支持。

应用程序的变化是非常快的，因为用户的需求总是在不停地发生变化，相比而言，一个好的数据库模式的生存期总是比一个应用程序的生存期要长。所以，数据库低层的安全模型对于实现整体的数据安全性具有更重大的意义。

VPD 机制实现面向数据的安全支持，提供数据安全与应用程序的相对独立性。在 VPD 机制的框架下，修改安全策略时，无须变更应用程序；修改应用程序时，也不必变更安全策略。

7.1.3　基于访问类型的控制的实施

在 7.1.2 节中已经介绍了 VPD 机制可以根据 SELECT、INSERT、UPDATE、DELETE 或 INDEX 等语句类型进行有针对性的访问控制。只是，还没有介绍这样的访问控制是如何实施的。在这一节里，以 INSERT、UPDATE 和 DELETE 语句为代表，介绍 VPD 机制基于语句类型的访问控制的实施方法。

首先，假设有如下的安全需求：

（1）允许用户查看所有的数据记录；

（2）只允许用户插入和更新他所在部门的数据记录；

（3）只允许用户删除他个人的数据记录。

为了实现以上需求，需要知道：哪个用户正在执行操作？该用户的用户名是什么？该用户所在部门的部门号是什么？可以通过环境参数来获取这些信息。不过，在这里，不关心通过环境参数来获取这些信息的具体方法。

我们姑且认为，通过调用 SYS_CONTEXT 函数就可以得到有关用户和部门的相关信息。即执行以下调用可以得到用户名信息：

```
sys_context("userenv","session_user")
```

执行以下调用可以得到用户所在部门的部门号：

```
sys_context("people_ctx","deptno")
```

1．INSERT/UPDATE访问控制的实施

为了实现“只允许用户插入和更新他所在部门的数据记录”的安全需求，需要为 INSERT 和 UPDATE 操作建立一个安全策略。下面的示例定义所需要的策略函数。

例 7.9　用户 SEC_MGR 执行以下操作，定义策略函数 dept_only：

```
CREATE OR REPLACE FUNCTION dept_only (
   p_schema      IN    VARCHAR2 DEFAULT NULL,
   p_objectIN    VARCHAR2 DEFAULT NULL)
   RETURN VARCHAR2
AS
BEGIN
   RETURN 'deptno = sys_context("people_ctx","deptno")';
END;
```

例 7.9 中的策略函数 dept_only 是在安全管理员模式 SEC_MGR 中建立的，这样可以借助不同的模式实现策略函数与数据的分离，有利于实现对安全策略的保护。

例 7.9 给策略函数的两个参数指定了默认值，都是 NULL，这便于在对参数进行赋值之前就可以对函数进行测试。下面的示例可以查看策略函数返回的谓词字符串。

例 7.10　用户 SEC_MGR 执行以下操作，查看策略函数 dept_only 返回的谓词字符串：

```
SELECT dept_only predicate FROM DUAL;
```

查询结果是：

```
PREDICATE
--------------------------------------------------------
deptno = sys_context('people_ctx','deptno')
```

下面的示例为 INSERT 和 UPDATE 操作创建一个安全策略，该安全策略创建后，只要用户在模式 SCOTT 中的表 PEOPLE 上执行插入或更新操作，系统就调用模式 SEC_MGR 中的策略函数 DEPT_ONLY 实施访问控制。

例 7.11　用户 SEC_MGR 执行以下操作，创建一个 INSERT/UPDATE 安全策略：

```
BEGIN
  DBMS_RLS.add_policy
              (object_schema      => 'SCOTT',
               object_name        => 'PEOPLE',
               policy_name        => 'PEOPLE_IU',
               function_schema    => 'SEC_MGR',
               policy_function    => 'Dept_Only',
               statement_types    => 'INSERT,UPDATE',
               update_check       => TRUE);
END;
```

下面的示例说明了例 7.11 中的 INSERT/UPDATE 安全策略创建之后，系统自动执行的对 UPDATE 语句的修改方法，这也是图 7.2 中显示的例子。

例 7.12　当用户执行以下的语句时：

```
update people
  set ename = '<NEW_VALUE>'
```

系统将自动把它修改为如下形式并执行：

```
update people
  set ename = '<NEW_VALUE>'
  where deptno = sys_context('people_ctx','deptno')
```

下面先来看一下执行 UPDATE 操作的例子。

例 7.13　用户 SCOTT 执行以下操作，查询目前表中有关用户和部门的数据情况：

```
SELECT username, deptno
    FROM people
   WHERE username < 'C';
```

查询结果是：

```
USERNAME            DEPTNO
------------------------- ------------
ALLEN                    30
BLAKE                    30
ADAMS                    20
```

例 7.14　设用户 SCOTT 所在部门的部门号为 20，他执行以下操作时，将有一条记录被更新：

```
UPDATE people
    SET username = 'GRIZZLY'
  WHERE username = 'ADAMS';
```

例 7.15　设用户 SCOTT 所在部门的部门号为 20，他执行以下操作时，将没有记录被更新：

```
UPDATE people
    SET username = 'BOZO'
  WHERE username = 'BLAKE';
```

当用户 SCOTT 执行例 7.15 中的语句时，系统自动把它修改为以下语句：

```
UPDATE people
    SET ename    = 'Bozo'
 WHERE ename    = 'BLAKE'
    AND deptno = SYS_CONTEXT ('people_ctx', 'deptno')
```

由于用户 SCOTT 所在部门的部门号为 20，所以实际执行的等价语句是：

```
UPDATE people
    SET ename    = 'Bozo'
 WHERE ename    = 'BLAKE'
    AND deptno = 20
```

由例 7.13 可知，用户 BLAKE 所在部门的部门号是 30，所以，表中不存在用户名为 BLAKE 且部门号为 20 的记录，因而，用户 SCOTT 执行例 7.15 中的操作时，没有记录被更新。

下面再来看一下执行 INSERT 操作的例子。

例 7.16　设用户 SCOTT 所在部门的部门号为 20，他执行以下操作时，将有一条记录被插入到表中：

```
INSERT INTO people
                (username, job, salary, deptno)
    VALUES ('KNOX', 'Clerk', '3000', 20);
```

例 7.17　设用户 SCOTT 所在部门的部门号为 20，他执行以下操作时，将没有记录被插入到表中：

```
INSERT INTO people
                (username, job, salary, deptno)
    VALUES ('ELLISON', 'CEO','90000', 30);
```

这里有一点需要注意：例 7.11 在创建 INSERT/UPDATE 安全策略时，给 UPDATE_CHECK 选项设置了 TRUE 值，所以，当用户执行插入操作时，系统把该安全策略实施到插入语句中，因而，例 7.17 中的插入操作没有成功。UPDATE_CHECK 选项的默认值是 FALSE，如果没有把它设置为 TRUE，系统将不对插入操作实施该安全策略，进而，例 7.17 中的插入操作将成功执行。

2．DELETE访问控制的实施

为了实现“只允许用户删除他个人的数据记录”的安全需求，需要为 DELETE 操作建立一个安全策略。下面的示例定义所需要的策略函数。

例 7.18　用户 SEC_MGR 执行以下操作，定义策略函数 user_only：

```
CREATE OR REPLACE FUNCTION user_only (
  p_schema     IN   VARCHAR2 DEFAULT NULL,
  p_objectIN   VARCHAR2 DEFAULT NULL)
  RETURN VARCHAR2
AS
BEGIN
  RETURN 'username = sys_context("userenv","session_user")';
END;
```

下面的示例创建一个 DELETE 安全策略。

例 7.19　用户 SEC_MGR 执行以下操作，创建一个 DELETE 安全策略：

```
BEGIN
  DBMS_RLS.add_policy
              (object_schema      => 'SCOTT',
               object_name        => 'PEOPLE',
               policy_name        => 'People_Del',
               function_schema    => 'SEC_MGR',
               policy_function    => 'user_only',
               statement_types    => 'DELETE');
END;
```

创建了例 7.19 中的 DELETE 安全策略后，用户将只能删除用户名等于他自己的数据记录。如下面的示例。

例 7.20　用户 SCOTT 执行以下操作，将删除且只删除用户名为 SCOTT 的数据记录：

```
DELETE FROM people;
```

用户 SCOTT 执行以下操作后，可以看到，SCOTT 的数据记录已被删除：

```
SELECT * FROM people
    WHERE username = 'SCOTT';
```

7.1.4　VPD安全防线

VPD 机制是数据库系统传统的安全防线之外的另一道安全防线。例如，如果想防止用户访问某张表中的任何记录，可以使用表的访问授权机制，撤销用户对该表的一切访问权限。

同时，可以在此基础上，通过 VPD 机制增设另一道安全防线，以增强系统的安全性。VPD 机制甚至可以防止拥有 DBA 角色的用户访问受保护的表。

在创建了如例 7.19 所示的 DELETE 安全策略的情况下，拥有 DBA 角色的用户 SYSTEM 也无法删除表中的数据记录，下面的示例说明了这一点。

例 7.21　由于用户 SYSTEM 拥有“SELECT ANY TABLE”特权，所以，他执行以下操作时，可以查询表中记录：

```
SELECT COUNT(*) FROM scott.people;
```

查询结果是：

```
COUNT(*)
------------
      14
```

但是，当他执行以下操作时，删除不了任何记录：

```
DELETE FROM scott.people;
```

当然，虽然 VPD 机制的控制对 DBA 是有效的，但是，DBA 有权执行更改策略函数、删除数据库表、停用 VPD 安全策略等操作，所以，DBA 不会因恶意用户滥用 VPD 机制而永远失去对数据库表的访问能力。

防止用户访问数据记录的最有效的办法就是设计一个返回恒假谓词的策略函数。例如，返回字符串“1=0”。下面的示例就是这样的函数。

例 7.22　用户 SEC_MGR 执行以下操作，定义一个产生恒假谓词的函数：

```
CREATE OR REPLACE FUNCTION no_records (
  p_schema      IN    VARCHAR2 DEFAULT NULL,
  p_objectIN    VARCHAR2 DEFAULT NULL)
  RETURN VARCHAR2
AS
BEGIN
  RETURN '1=0';
```

```
END;
```

假设用户 SCOTT 执行以下操作创建了表 people_ro：

```
CREATE TABLE people_ro AS SELECT * FROM emp;
```

下面的示例创建一个安全策略，使 people_ro 成为一张只允许读的表。

例 7.23　用户 SEC_MGR 执行以下操作，创建一个安全策略，禁止所有用户对表 people_ro 进行插入、更新或删除操作：

```
BEGIN
  DBMS_RLS.add_policy
      (object_schema      => 'SCOTT',
       object_name        => 'PEOPLE_RO',
       policy_name        => 'PEOPLE_RO_IUD',
       function_schema    => 'SEC_MGR',
       policy_function    => 'No_Records',
       statement_types    => 'INSERT,UPDATE,DELETE',
       update_check       => TRUE);
END;
```

下面的示例表明表 people_ro 具有只允许读的性质。

例 7.24　用户 SCOTT 执行以下操作，可以查询到表 people_ro 的数据记录：

```
SELECT COUNT (*) FROM people_ro;
```

查询结果是：

```
COUNT(*)
------------
      14
```

用户 SCOTT 执行以下操作，但更新不了任何记录：

```
UPDATE people_ro
    SET ename = NULL;
```

用户 SCOTT 执行以下操作，但删除不了任何记录：

```
DELETE FROM people_ro;
```

用户 SCOTT 执行以下操作，但无法插入记录：

```
INSERT INTO people_ro (ename) VALUES ('KNOX');
```

7.1.5　免受VPD机制控制的措施

在正常情况下，即使是 DBA 和数据的拥有者，也摆脱不了 VPD 机制安全策略的控制。有时，这会给数据库的维护工作造成障碍，如妨碍数据备份工作的顺利开展。针对

这样的问题，VPD 机制提供一个免受控制的特权，即“EXEMPT ACCESS POLICY”特权，获得该特权的用户，在访问数据库数据时，可以摆脱 VPD 机制所有策略函数的限制，像根本就没有任何策略函数存在一样。

下面的示例演示：在获得“EXEMPT ACCESS POLICY”特权前后，用户 SYSTEM 进行数据访问操作的效果。

例 7.25　正常情况下，用户 SYSTEM 执行以下删除操作，但删除不了任何数据记录：

```
DELETE FROM scott.people_ro;
```

用户 SYSTEM 执行以下操作，给自己授予免受 VPD 控制的特权：

```
GRANT EXEMPT ACCESS POLICY TO SYSTEM;
```

用户 SYSTEM 再执行以下删除操作时，便能删除表中的所有记录：

```
DELETE FROM scott.people_ro;
```

以上操作只是出于演示所需，为了不破坏表中数据，用户 SYSTEM 可以执行以下回滚操作：

```
ROLLBACK ;
```

“EXEMPT ACCESS POLICY”是非常强大的一个特权，授权时，需要特别慎重。下面的示例可以查询系统中获得了该特权的用户的信息。

例 7.26　用户 SYSTEM 执行以下操作，了解哪些用户获得了免受控制的特权：

```
SELECT grantee
    FROM dba_sys_privs
   WHERE PRIVILEGE = 'EXEMPT ACCESS POLICY';
```

查询结果是：

```
GRANTEE
------------------------------
SYSTEM
```

7.1.6　面向敏感字段的VPD功能

VPD 机制可以针对表的敏感字段启动访问控制功能，也就是说，可以把一张表中的某些字段确定为敏感字段，并告诉系统：当且仅当用户对表中的敏感字段进行访问时，才启动相应的 VPD 安全策略进行访问控制，如果用户只对表中的其他字段进行访问，则不必启动 VPD 安全策略的访问控制功能。

例如，对于员工信息表而言，可以确定工资字段为敏感字段，利用 VPD 机制面向敏感字段的访问控制特性，可以允许任何用户查看员工信息表中所有记录中除工资字段

以外的所有信息，而只允许用户查看自己的工资信息。

利用例 7.18 定义的 user_only 策略函数，下面的示例创建一个安全策略来实现这样的安全需求。

例 7.27　用户 SEC_MGR 执行以下操作，创建一个面向 SALARY 敏感字段的 SELECT 安全策略：

```
BEGIN
  DBMS_RLS.add_policy
                    (object_schema      => 'SCOTT',
                     object_name        => 'PEOPLE',
                     policy_name        => 'people_sel_sal',
                     function_schema    => 'SEC_MGR',
                     policy_function    => 'user_only',
                     statement_types    => 'SELECT',
                     sec_relevant_cols  => 'SALARY');
END;
```

例 7.27 中的 SEC_RELEVANT_COLS 选项的值确定了 SALARY 字段为敏感字段，也确定了例 7.27 中所创建的安全策略 people_sel_sal 为面向敏感字段的安全策略，当且仅当查询操作涉及 SALARY 字段时，该安全策略实施控制作用。

在下面的示例中，安全策略 people_sel_sal 不起作用，因为用户没有查询敏感字段。

例 7.28　用户 SCOTT 执行以下操作，查询人员姓名信息，不涉及敏感字段：

```
SELECT username FROM people
   WHERE ROWNUM <= 5;
```

查询结果是：

```
USERNAME
----------
SMITH
ALLEN
WARD
JONES
MARTIN
```

在下面的示例中，安全策略 people_sel_sal 发挥作用，因为用户的查询涉及敏感字段。

例 7.29　用户 SCOTT 执行以下操作，查询人员姓名和工资信息，涉及敏感字段：

```
SELECT username, salary FROM people;
```

查询结果是：

```
USERNAME          SALARY
------------------------- -------------
SCOTT                3000
```

当查询操作涉及敏感字段时，例 7.27 定义的安全策略只允许用户查询自己的记录信息。可以对该安全策略做一点修改，以便允许用户查询其他的记录信息，但屏蔽掉那些记录中的敏感字段信息。下面的示例给出的就是出于这样的考虑而修改过的安全策略的描述。

例 7.30　用户 SEC_MGR 执行以下操作，创建新的 people_sel_sal 安全策略：

```
BEGIN
  -- 移除现有安全策略
  DBMS_RLS.drop_policy
              (object_schema          => 'SCOTT',
               object_name            => 'PEOPLE',
               policy_name            => 'people_sel_sal');
  -- 重新登记安全策略，增加 SEC_RELEVANT_COLS_OPT 选项
  DBMS_RLS.add_policy
              (object_schema          => 'SCOTT',
               object_name            => 'PEOPLE',
               policy_name            => 'people_sel_sal',
               function_schema        => 'SEC_MGR',
               policy_function        => 'user_only',
               statement_types        => 'SELECT',
               sec_relevant_cols      => 'SALARY',
               sec_relevant_cols_opt  => DBMS_RLS.all_rows);
END;
```

例 7.30 中的 SEC_RELEVANT_COLS_OPT 选项的值表示允许查询所有记录的数据，使那些在策略函数的限定下查询不到的记录也能够被查询出来，不过，那些记录的敏感字段的数据都将是不可见的。只有在策略函数的限定下也能查询到的记录的敏感字段的数据才是可见的。下面的示例演示在新的安全策略下的一个查询情况。

例 7.31　用户 SCOTT 执行以下操作，查询部门号为 20 的员工名和工资信息，涉及敏感字段：

```
SELECT username, salary FROM people
   WHERE deptno = 20;
```

查询结果是：

```
USERNAME          SALARY
------------------------- -------------
SMITH
JONES
SCOTT                3000
ADAMS
FORD
```

这个示例的查询操作涉及 SALARY 敏感字段，在策略函数 user_only 的限定下，该操作只能查询到 SCOTT 的记录信息，由于 SEC_RELEVANT_COLS_OPT 选项的作用，该操作也查询出了其他记录信息，但是，那些记录的 SALARY 字段的值都是不可见的，只有 SCOTT 的记录的 SALARY 字段的值是可见的。

面向字段敏感性的 VPD 功能对于支持私密性安全需求的实现具有重要意义，它允许把敏感信息和非敏感信息存放在一起，同时能够确保敏感信息不会泄露给无关的用户。

7.2　基于标签的安全机制

基于标签的安全机制的基本思想是引入一种称为标签（Label）的安全属性，给主体和客体都设定这种安全属性。即，给一个主体分配一个标签，给一个客体分配一个标签，当一个主体请求对一个客体进行访问时，把主体的标签与客体的标签进行对比，以对比的结果为依据判定是否允许进行相应的访问。

著名的贝尔－拉普杜拉模型研究的就是一种经典的基于标签的安全系统。在信息安全系统的发展进程中，大量的强制访问控制机制都是基于标签的安全机制的。Oracle 吸收了传统信息安全系统中的标签思想，并对它进行了发展和扩充，设计并实现了面向数据库记录的基于标签的安全技术，简称为 OLS 技术（OLS，Oracle Label Security）。本节将围绕 OLS 技术来探讨运用标签增强数据库系统的安全性的方法。

7.2.1　标签的基本构成

传统的标签由两类元素组成，其中一类元素称为等级，另一类元素称为类别。一个传统的标签可以表示为以下的二元组：

（等级，类别）

其中，等级是有序的，任意两个等级之间是可以比较大小的。类别通常与集合相对应，类别之间没有大小之分。

例 7.32　信息的敏感级别可以划分为绝密、机密、秘密和非密，部队的军种可以划

分为海军、空军和陆军，基于这样的背景，可以为部队的信息给出标签的定义，其中，等级元素的取值可以是绝密、机密、秘密或非密，类别元素的取值可以是由海军、空军和陆军组成的任意集合。以下是几个标签示例。

标签 1：（绝密，{海军}）

标签 2：（机密，{陆军}）

标签 3：（机密，{海军，空军}）

在例 7.32 的约定之下，可以定义出很多标签，以上只是随意地给出了其中的几个。其中，标签 1 表示海军中的绝密信息，标签 2 表示陆军中的机密信息，标签 3 表示海军和空军的机密信息。

OLS 在传统标签思想的基础之上引入了分组的思想，用三类元素来定义标签，即等级、类别和组别。这样，OLS 的一个标签可以表示为如下的三元组：

（等级，类别，组别）

例 7.33　在例 7.32 的基础上，可以根据地理分布定义以下分组：美国、纽约和夏威夷，以下是几个 OLS 标签的示例：

标签 4：（绝密，{海军}，纽约）

标签 5：（绝密，{海军}，夏威夷）

标签 6：（绝密，{海军}，美国）

在例 7.33 中，标签 4 表示在纽约的海军中的绝密信息，标签 5 表示在夏威夷的海军中的绝密信息，标签 6 表示在美国的海军中的绝密信息。

注意：在美国、纽约和夏威夷这三个组中，显然，美国包含纽约，也包含夏威夷，因此，组别之间可以形成层次关系。但是，类别之间没有层次关系。

OLS 的安全机制为应用提供了灵活的标签定义支持。OLS 规定标签中的等级元素必须取一个值，而且只能取一个值，但是，标签中的类别和组别元素可以取一个值，也可以取多个值，还可以取 0 个值（即不取值）。

例 7.34　在 OLS 的安全机制中，以下表示都是有效的标签：

（绝密）

（绝密：海军）

（绝密：海军，空军）

（绝密：海军，空军：纽约）

（机密：海军：纽约，夏威夷）

其中，以下是 OLS 标签的一般表示形式：

（等级：类别：组别）

7.2.2　基于标签的数据库访问控制

OLS 技术提供记录级的数据库访问控制支持。就客体而言，它为数据库表中的记录分配标签；就主体而言，它为用户分配标签。记录的标签表示记录中的数据所具有的敏感程度，用户的标签表示用户所拥有的对记录进行访问的授权。

例 7.35　若给某数据库记录分配以下标签，则表示该记录中的信息是有关纽约的海军中的机密信息的：

（机密：海军：纽约）

若给某用户分配以上标签，则表示该用户获得了对有关纽约的海军中的机密记录信息进行访问的授权。

在例 7.35 中，具体的访问权限由与标签相关的授权操作确定。在给用户分配标签时，要结合标签中的类别和组别进行相应的授权，授权操作可以针对标签中的类别和组别指定读（READ）和写（WRITE）等权限。读权限表示允许用户对相应类别或组别的记录执行 SELECT 操作，写权限表示允许用户对相应类别或组别的记录执行 INSERT、UPDATE 和 DELETE 操作。

OLS 机制根据用户的标签和记录的标签对访问请求进行判定，图 7.3 描绘了用户、数据、标签和访问控制的基本原理。

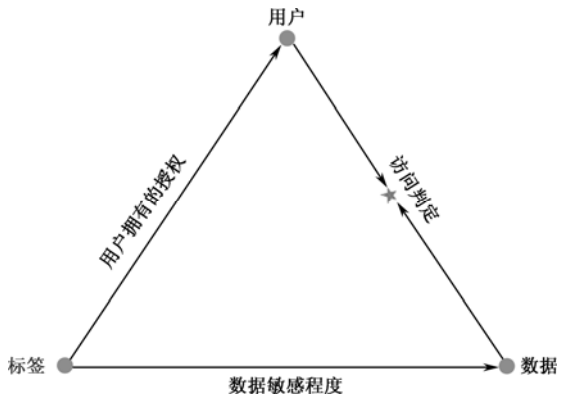


图 7.3　用户、数据、标签和访问控制的基本原理

1．基于标签的读访问控制规则

当用户请求对数据库记录进行读访问时，OLS 机制根据以下规则进行访问判定：

（1）用户标签中的等级必须大于或等于记录标签中的等级；

（2）用户标签中的组别至少必须包含记录标签中的一个组别，并且，用户拥有对该组别的读访问授权；

（3）用户标签中的类别必须包含记录标签中的所有类别。

在规则（2）中，当用户标签中的某个组别是记录标签中的某个组别的父组别时，条件是满足的。例如，用户标签的组别中含有美国，记录标签的组别中含有纽约，因为美国这个组别显然是纽约这个组别的父组别，所以，此时规则（2）的条件得到满足。

图 7.4 描绘了基于标签的读访问的判定过程。

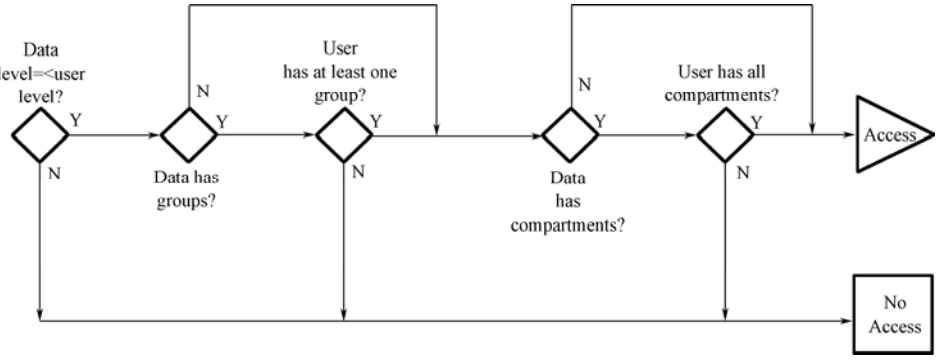


图 7.4　基于标签的读访问的判定过程

2．基于标签的写访问控制规则

当用户请求对数据库记录进行写访问时，OLS 机制根据以下规则进行访问判定：

（1）记录标签中的等级必须大于或等于用户的最小等级；

（2）记录标签中的等级必须小于或等于用户会话标签中的等级；

（3）用户会话标签中的组别必须至少包含记录标签中的一个组别，并且，用户拥有对该组别的写访问授权；

（4）用户会话标签中的类别必须包含记录标签中的所有类别；

（5）如果记录标签中的组别不取值，用户必须对记录标签中的所有类别拥有写访问授权。

以上规则中用到了用户会话标签这个术语。事实上，用户总是在某个会话中对数据库记录进行访问的，用户在该会话中分配到的标签就称为该用户的会话标签。实际上，在访问控制判定中，所说的用户标签都是指用户会话标签。

图 7.5 描绘了基于标签的写访问请求的判定过程。

7.2.3　基于标签的安全机制的实现

应用 OLS 技术实现基于标签的安全机制，需要执行以下操作。

（1）创建安全策略：此处的安全策略指支持基于标签的安全性的安全策略，它描述基于标签的安全性的相关成分和规则。在这里，创建一个安全策略的意义相当于创建一个容器，该容器将用于容纳标签信息、用户授权信息和待保护的数据库对象信息。

（2）定义标签元素：给出构成标签的等级、类别和组别等元素的定义。

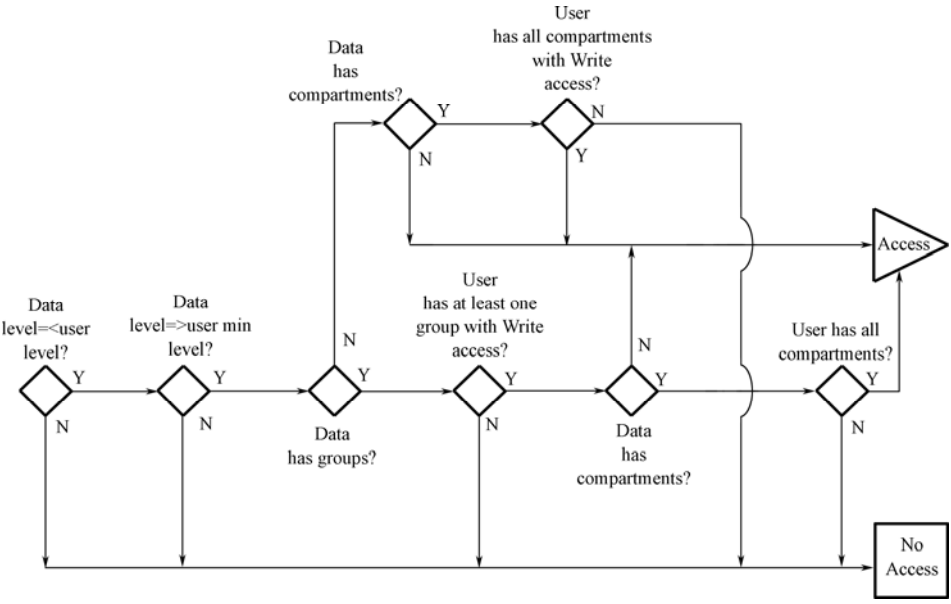


图 7.5　基于标签的写访问的判定过程

（3）创建实际使用的标签：结合应用系统的安全策略，利用已定义的标签元素创建将要使用的各种标签。

（4）把安全策略（标签）实施到数据库表或模式中：给相关的数据库表增加一个用于描述标签的字段，并建立相应的基础支持机制，以支持基于标签的记录级数据库安全性，同时定义安全策略的实施方法。

（5）为用户建立基于标签的访问授权：把标签指派给用户，以确定哪些主体最终可以访问哪些客体。这里所说的用户可以是一个实际的用户，也可以是一个用户群，安全机制将根据指派给用户的标签与数据库记录的标签的比较结果进行访问仲裁。

下面将结合示例给以上的各个步骤赋予实际的意义，与此同时，对标签的概念和基于标签的安全机制的基本原理进行进一步的具体阐述。

7.2.4　OLS安全策略的创建

7.2.3 节的内容告诉我们，为了应用 OLS 技术，实现基于标签的数据库安全支持，第一步要完成的工作是建立 OLS 安全策略。标签的定义及基于标签的访问控制授权是在 OLS 安全策略中描述的，在一个 OLS 安全策略中，可以建立一套特定的标签定义方法和基于标签的访问控制授权方法。

在传统的基于标签的典型安全系统中，如在实现贝尔–拉普杜拉模型的安全操作系统中，或在可信 Oracle 数据库系统中，整个系统必须采用一套统一的标签定义方法和基于标签的访问控制授权方法。也就是说，在这样的传统系统中，必须定义一套在整个系

统范围内适用的统一标签，整个系统范围内的访问控制必须以这一套统一的标签为基础进行授权和判定。

OLS 机制与传统系统的标签支持机制不同，它允许用户建立多个 OLS 安全策略，在不同的 OLS 安全策略中，用户可以定义不同的标签和进行不同的访问控制授权。因此，在 OLS 的支持下，用户可以根据不同应用的实际需要创建不同的 OLS 安全策略，从而为各种应用进行量身定制的标签定义和访问控制授权，实现灵活的面向应用的基于标签的记录级数据库安全支持。

既然 OLS 机制提供的是面向应用的数据库安全支持，在对它的思想和原理进行深入的讨论之前，首先建立一个应用示例。

例 7.36　设有一个简单的企业信息发布应用系统，其主要作用是给企业员工发布信息，基本要求是可以给不同类型的员工发布不同的信息，每个员工只能看到他应该看的信息，看不到他不该看的信息。

用一张数据库表来存储发布的信息，在数据库模式 SCOTT 中创建数据库表，利用数据库模式 SEC_MGR 进行安全管理。用户 SCOTT 执行以下操作创建信息发布表：

```
CREATE TABLE announcements    (MESSAGE        VARCHAR2(4000));
```

每个 OLS 安全策略需要有一个用于描述标签的字段与其对应，创建 OLS 安全策略时，需要为该标签字段确定一个名称。在一个数据库系统中，标签字段的名称必须是唯一的，不同安全策略中的标签字段的名称不能相同。

创建安全策略需要使用管理员身份，在 OLS 机制中，默认的管理员是 LBACSYS。下面的示例创建一个 OLS 安全策略。

例 7.37　设安全策略的名称为 ESBD，描述标签的字段的名称为 ROWLABEL，管理员 LBACSYS 执行以下存储过程创建一个 OLS 安全策略：

```
BEGIN
    sa_sysdba.create_policy
                    (policy_name        => 'ESBD',
                     column_name        => 'rowlabel');
END;
```

在 OLS 机制中，创建了一个安全策略之后，系统自动为该安全策略创建一个数据库管理员角色，角色的名称为策略名称之后添上“_DBA”，例 7.37 创建的安全策略为 ESBD，系统为该安全策略创建的相应管理员角色便是 ESBD_DBA。

对安全策略 ESBD 的操作需要管理员角色 ESBD_DBA 的权限，为使用户 SEC_MGR 能够对策略 ESBD 进行操作，可把角色 ESBD_DBA 的权限授予该用户，下面的示例可达到这个目的。

例 7.38　管理员 LBACSYS 执行以下操作，把对安全策略 ESBD 进行管理的权限授予用户 SEC_MGR：

```
GRANT esbd_dba TO sec_mgr;
```

为了在安全策略管理中支持职权分离的原则，OLS 并没有把管理策略 ESBD 的所有特权都授予角色 ESBD_DBA。例如，执行 OLS 的以下程序还需要各自相关的程序执行权限：

```
sa_components
sa_label_admin
sa_user_admin
char_to_label
```

通过后面内容的学习还会看到，在标签的定义和访问控制授权中，需要执行这四个程序。下面的示例把执行这些程序的权限授予用户 SEC_MGR。

例 7.39　管理员 LBACSYS 执行以下操作，把执行上面列出的四个程序的权限授予用户 SEC_MGR：

```
GRANT EXECUTE ON sa_components    TO sec_mgr;
GRANT EXECUTE ON sa_label_admin    TO sec_mgr;
GRANT EXECUTE ON sa_user_admin    TO sec_mgr;
GRANT EXECUTE ON char_to_label    TO sec_mgr;
```

7.2.5　标签的等级元素的创建与应用

创建了 OLS 安全策略之后，便可以在安全策略中对标签进行定义。通过前面的学习我们已经知道，OLS 的标签由等级、类别和组别三种元素构成，其中，等级元素必须取一个值，而且只能取一个值；类别元素可以取 0 个、1 个或多个值；组别元素也可以取 0 个、1 个或多个值。

为了能够遵循由简到繁、循序渐进的学习过程，首先考察仅由等级元素构成的标签，即类别元素和组别元素都取 0 个值的标签。

1．等级元素的定义

标签中的等级元素用于表示信息或用户的安全等级。等级元素的描述由三部分内容构成：一个短名称、一个长名称和一个等级数值。等级数值用于表示等级的高低，数值越大，等级越高；数值越小，等级越低。以字符串的形式引用等级元素时，使用的是等级元素的短名称。

例 7.40　针对 ESBD 应用示例，把企业的员工划分为三个等级，最高等级由企业的高层人员构成，包括首席执行官（CEO）、首席财务官（CFO）、首席信息官（CIO）和

执行副总裁（CVP）等，第二等级由企业的管理人员构成，最低等级由企业的普通员工构成。用户 SEC_MGR 执行以下操作定义三个相应的等级：

```
BEGIN
    -- 创建与企业高层人员相对应的最高等级
    sa_components.create_level
                    (policy_name     => 'ESBD',
                     long_name       => 'Executive Staff',
                     short_name      => 'EXEC',
                     level_num       => 9000);
    -- 创建与管理人员相对应的等级
    sa_components.create_level
                    (policy_name     => 'ESBD',
                     long_name       => 'Manager',
                     short_name      => 'MGR',
                     level_num       => 8000);
    -- 创建与普通员工相对应的等级
    sa_components.create_level
                    (policy_name     => 'ESBD',
                     long_name       => 'Employee',
                     short_name      => 'EMP',
                     level_num       => 7000);
END;
```

由例 7.40 可知，定义标签元素需要执行 SA_COMPONENTS 程序，等级元素的定义由存储过程 CREATE_LEVEL 完成。等级元素中的等级数值由参数 LEVEL_NUM 进行赋值。等级元素通过短名称来引用，因此，例 7.40 定义了 EXEC、MGR 和 EMP 三个等级元素，这三个等级元素的等级数值分别为 9000、8000 和 7000。

因为 9000 > 8000 > 7000（此处的“>”表示数值的大小），所以 EXEC > MGR > EMP（此处的“>”表示等级的高低）。

2．标签的创建

定义了标签的元素之后，就可以利用已定义的标签元素来创建标签，如下面的示例所示。

例 7.41　用户 SEC_MGR 执行以下操作创建三个标签：

```
BEGIN
    -- 创建企业高层人员标签
    sa_label_admin.create_label
```

```
                    (policy_name      => 'ESBD',
                     label_tag        => 1,
                     label_value      => 'EXEC');
    -- 创建企业管理层标签
    sa_label_admin.create_label
                    (policy_name      => 'ESBD',
                     label_tag        => 2,
                     label_value      => 'MGR');
    -- 创建普通员工标签
    sa_label_admin.create_label
                    (policy_name      => 'ESBD',
                     label_tag        => 3,
                     label_value      => 'EMP');
END;
```

创建标签时，必须为每个标签指派一个唯一的标签标识号。标签标识号的唯一性要求是：在一个数据库系统的所有 OLS 安全策略的所有标签中，任意两个标签的标签标识号都不能相等。把 OLS 安全策略实施到数据库表中时，标签标识号将存入到表的标签字段中，用于标识相应的标签。

标签标识号通过参数 LABEL_TAG 进行赋值。标签元素的值通过参数 LABEL_VALUE 赋给标签。标签标识号的大小与标签的安全等级无关，在例 7.41 中，三个标签的标签标识号的大小顺序正好与标签的安全等级的高低顺序相反，即标签标识号最小的标签的安全等级最高，这是为了说明问题而故意设计的。

3．安全策略的实施

至此，已经创建了 OLS 安全策略、定义了标签元素、创建了标签，接着，就可以把安全策略实施到数据库表中了。前面创建的数据库表是一张空表，为了演示的需要，先通过下面的例子给它填上一些信息。

例 7.42　用户 SCOTT 执行以下操作给数据库表插入三条记录：

```
INSERT INTO announcements
        VALUES ('This message is only for the Executive Staff.');
INSERT INTO announcements
        VALUES ('All Managers: employee compensation announcement...');
INSERT INTO announcements
        VALUES ('This message is to notify all employees...');
```

在例 7.42 中，按照插入记录的先后顺序，第一条记录的信息是为企业高层人员提供的，第二条记录的信息是为管理层人员提供的，第三条记录的信息是为普通员工提供的。

下面的例子把安全策略实施到数据库表中。

例 7.43　用户 SEC_MGR 执行以下操作，把安全策略 ESBD 实施到数据库表 ANNOUNCEMENTS 中：

```
BEGIN
    sa_policy_admin.apply_table_policy
                        (policy_name      => 'ESBD',
                         schema_name      => 'SCOTT',
                         table_name       => 'ANNOUNCEMENTS',
                         table_options    => 'NO_CONTROL');
END;
```

把一个安全策略实施到一张数据库表中，系统将为该数据库表创建一个专用于描述标签信息的字段，这里用于描述标签的字段的名称为 ROWLABEL。新创建的标签字段的字段值还没有定义，在这种情况下，如果让 OLS 对相应的表实施访问控制，将无法正常访问到表中的记录信息，因为 OLS 不会返回标签字段值没有定义或为 NULL 的任何记录。为此，例 7.43 设置了 NO_CONTROL 选项，使 OLS 暂时不对相应的表实施访问控制，以便能对其中的记录进行访问，并为它们的标签字段设置适当的值。

下面的例子给数据库表中的标签字段赋值。

例 7.44　用户 SCOTT 执行以下操作，给数据库表中所有记录的标签字段赋予最低等级 EMP 的标签值：

```
UPDATE scott.announcements
    SET ROWLABEL = char_to_label ('ESBD', 'EMP');
```

例 7.45　用户 SCOTT 执行以下操作，给数据库表中与企业管理层对应的记录的标签字段赋予管理层等级 MGR 的标签值：

```
UPDATE scott.announcements
    SET ROWLABEL = char_to_label ('ESBD', 'MGR')
  WHERE UPPER (MESSAGE) LIKE '%MANAGE%';
```

例 7.46　用户 SCOTT 执行以下操作，给数据库表中与企业高层对应的记录的标签字段赋予高层等级 EXEC 的标签值：

```
UPDATE scott.announcements
    SET ROWLABEL = char_to_label ('ESBD', 'EXEC')
  WHERE UPPER (MESSAGE) LIKE '%EXECUTIVE%';
```

通过以上三个例子，给数据库表中记录的标签字段都赋予了有效的值，这意味着表中的记录都具有了有效的标签属性。如果再对用户进行基于标签的访问授权，就可以对

数据库表实施基于标签的访问控制了。

4．基于标签的访问授权

基于标签的访问授权实际上就是为用户分配标签，但在标签的分配形式上，不一定把标签直接分配给一个实际的数据库用户，有可能把标签分配给某用户群，还有可能把标签分配给应用系统、IP 地址域或其他相关对象。

如果将标签直接分配给一个实际的数据库用户，则该用户登录后便自动获得相应的授权。如果将标签分配给其他对象，则需要在进行访问前把标签映射为数据库会话标签，以便在该会话中的用户获得相应的授权。

下面的例子把标签分配给用户群。

例 7.47　用户 SEC_MGR 执行以下操作，分别把标签分配给与普通员工、企业管理层、企业高层相对应的三个用户群：

```
BEGIN
    sa_user_admin.set_user_labels
                    (policy_name          => 'ESBD',
                     user_name            => 'ALL_EMPLOYEES',
                     max_read_label       => 'EMP');
    sa_user_admin.set_user_labels
                     (policy_name         => 'ESBD',
                      user_name           => 'ALL_MANAGERS',
                      max_read_label      => 'MGR');
    sa_user_admin.set_user_labels
                       (policy_name       => 'ESBD',
                        user_name         => 'ALL_EXECS',
                        max_read_label    => 'EXEC');
  END;
```

由于上面这个例子把标签分配给了用户群，为了使用户在进行数据库访问时获得相应的授权，必须在访问前把分配给用户群的标签映射为数据库会话标签。下面的示例把用户群标签映射为数据库会话标签。

例 7.48　用户 SEC_MGR 执行以下操作，把分配给用户群 ALL_EMPLOYEES 的标签映射为数据库会话标签：

```
BEGIN
  sa_session.set_access_profile ('ESBD',
                                 'ALL_EMPLOYEES');
END;
```

为了执行如例 7.48 所示的把一个标签映射为数据库会话标签的操作，执行该操作的用户必须拥有 PROFILE_ACCESS 特权，下面的例子给出了获得该特权的方法。

例 7.49　用户 SEC_MGR 执行以下操作，把 PROFILE_ACCESS 特权分配给用户 SEC_MGR：

```
BEGIN
  sa_user_admin.set_user_privs
                    (policy_name      => 'ESBD',
                     user_name        => 'SEC_MGR',
                     PRIVILEGES       => 'PROFILE_ACCESS');
END;
```

例 7.49 的操作应该在例 7.48 的操作之前执行，这样，例 7.48 的操作才能成功地执行，因为例 7.48 的操作为用户 SEC_MGR 分配执行例 7.48 的操作所需要的特权。

5．基于标签的访问控制演示

前面介绍了创建安全策略、定义标签元素、创建有效标签、实施安全策略、分配用户标签等方法。运用这些方法，完成了基于标签的访问授权之后，已经基本准备好使用 OLS 机制进行基于标签的数据库记录访问控制了。

把安全策略实施到数据库表中时，通过设置策略实施选项，使 OLS 机制暂时不进行访问控制，现在，要改变策略实施选项，使 OLS 机制开始进行访问控制。为了改变策略的实施选项，首先要移去安全策略，然后再次实施，下面的示例完成这项工作。

例 7.50　用户 SEC_MGR 执行以下操作，激活 OLS 机制的基于标签的读访问控制功能：

```
BEGIN
  sa_policy_admin.remove_table_policy
                    (policy_name     => 'ESBD',
                     schema_name     => 'SCOTT',
                     table_name      => 'ANNOUNCEMENTS');
  sa_policy_admin.apply_table_policy
                   (policy_name      => 'ESBD',
                    schema_name      => 'SCOTT',
                    table_name       => 'ANNOUNCEMENTS',
                    table_options    => 'READ_CONTROL');
END;
```

下面的例子演示基于标签的访问控制效果。

例 7.51　用户 SEC_MGR 执行以下操作，首先把分配给普通员工群的标签映射为数据库会话标签，然后对信息进行查询：

```
BEGIN
    sa_session.set_access_profile ('ESBD',
                                   'ALL_EMPLOYEES');
END;
SELECT MESSAGE FROM scott.announcements;
```

查询结果如下：

```
This message is to notify all employees...
```

例 7.52　用户 SEC_MGR 执行以下操作，首先把分配给企业管理层员工群的标签映射为数据库会话标签，然后对信息进行查询：

```
BEGIN
  sa_session.set_access_profile ('ESBD',
                                 'ALL_MANAGERS');
END;
SELECT MESSAGE FROM scott.announcements;
```

查询结果如下：

```
All Managers: employee compensation announcement...
This message is to notify all employees...
```

例 7.53　用户 SEC_MGR 执行以下操作，首先把分配给企业高层员工群的标签映射为数据库会话标签，然后对信息进行查询：

```
BEGIN
  sa_session.set_access_profile ('ESBD',
                                 'ALL_EXECS');
END;
SELECT MESSAGE FROM scott.announcements;
```

查询结果如下：

```
This message is only for the Executive Staff.
All Managers: employee compensation announcement...
This message is to notify all employees...
```

7.2.6　标签的类别元素的创建与应用

我们已经学习了以等级元素为例的标签的定义和应用方法，现在可以开始学习类别元素的定义方法，并把它用到标签之中，从而扩充前面学过的标签的内容。

可以对信息进行分类，以每类信息为一个部分，把信息划分为若干个部分，使各个

部分之间具有相互隔离的关系，用这样的分类方法划分出的类别可以作为标签中的类别元素。类别之间没有大小或高低的关系，也没有层次关系。

在我们的示例应用中，可以把一个企业划分为销售部、开发部和内部系统支持部三个部分，以每个部分所对应的信息作为一个类别，便得到三个类别。下面就以这三个类别为例，学习标签的类别元素的有关知识。

1．类别元素的定义

下面的例子定义三个类别元素。

例 7.54　用户 SEC_MGR 执行以下操作，定义产品销售、产品开发和内部支持三个类别元素：

```
BEGIN
  sa_components.create_compartment
                        (policy_name     => 'ESBD',
                         long_name       => 'Product Sales',
                         short_name      => 'SALES',
                         comp_num        => 1000);
  sa_components.create_compartment
                        (policy_name     => 'ESBD',
                         long_name       => 'Product Development',
                         short_name      => 'DEV',
                         comp_num        => 100);
  sa_components.create_compartment
                        (policy_name     => 'ESBD',
                         long_name       => 'Internal Support',
                         short_name      => 'IS',
                         comp_num        => 10);
END;
```

从例 7.54 可以看出，与等级元素的描述相似，类别元素的描述也包含三部分内容：短名称、长名称和类别号，分别由参数 SHORT_NAME、LONG_NAME 和 COMP_NUM 定义。以字符串形式对类别的引用使用类别的短名称。因为类别之间没有大小或高低的关系，所以，类别号的大小并不表示类别之间的大小或高低。

2．含类别元素的标签的创建

可以把 7.2.5 节中定义的等级元素和例 7.54 定义的类别元素组合起来，创建包含等级和类别两种元素的标签。下面的例子给出这样的标签的创建方法。

例 7.55　用户 SEC_MGR 执行以下操作，创建一个与企业高层等级对应的包含三个

类别元素的标签：

```
BEGIN
  sa_label_admin.create_label
                (policy_name      => 'ESBD',
                 label_tag        => 10,
                 label_value      => 'EXEC:SALES,DEV,IS');
END;
```

例 7.55 中，由参数 LABEL_VALUE 定义的标签值表示：该标签的等级是 EXEC，它包含 SALES、DEV 和 IS 三个类别。

例 7.56　用户 SEC_MGR 执行以下操作，创建一个与企业管理层等级对应的包含一个类别元素的标签：

```
BEGIN
  sa_label_admin.create_label
                (policy_name      => 'ESBD',
                 label_tag        => 20,
                 label_value      => 'MGR:SALES');
  sa_label_admin.create_label
                (policy_name      => 'ESBD',
                 label_tag        => 25,
                 label_value      => 'MGR:DEV');
END;
```

例 7.57　用户 SEC_MGR 执行以下操作，创建一个与普通员工等级对应的包含一个类别元素的标签：

```
BEGIN
  sa_label_admin.create_label
                (policy_name      => 'ESBD',
                 label_tag        => 30,
                 label_value      => 'EMP:SALES');
  sa_label_admin.create_label
                (policy_name      => 'ESBD',
                 label_tag        => 35,
                 label_value      => 'EMP:DEV');
  sa_label_admin.create_label
                (policy_name      => 'ESBD',
                 label_tag        => 39,
                 label_value      => 'EMP:IS');
```

```
END;
```

3．基于类别标签的授权

这里所说的类别标签是指包含类别元素但未包含组别元素的标签，基于类别标签的授权就是把类别标签分配给用户。在前面已经学习了给用户分配标签的方法，尽管那时的标签只包含等级元素，现在的标签增加了类别元素，但是，标签的分配方法还是一样的。从下面的例子可以很容易地看到这一点。

例 7.58　用户 SEC_MGR 执行以下操作，给销售经理和开发经理这两个用户群各分配一个标签：

```
BEGIN
  sa_user_admin.set_user_labels
              (policy_name          => 'ESBD',
               user_name            => 'SALES_MANAGERS',
               max_read_label       => 'MGR:SALES');
  sa_user_admin.set_user_labels
              (policy_name          => 'ESBD',
               user_name            => 'DEV_MANAGERS',
               max_read_label       => 'MGR:DEV');
END;
```

例 7.59　用户 SEC_MGR 执行以下操作，给产品销售员工、产品开发员工和内部支持员工这三个用户群各分配一个标签：

```
BEGIN
  sa_user_admin.set_user_labels
              (policy_name          => 'ESBD',
               user_name            => 'SALES_EMPLOYEES',
               max_read_label       => 'EMP:SALES');
  sa_user_admin.set_user_labels
              (policy_name          => 'ESBD',
               user_name            => 'DEV_EMPLOYEES',
               max_read_label       => 'EMP:DEV');
  sa_user_admin.set_user_labels
              (policy_name          => 'ESBD',
               user_name            => 'INTERNAL_EMPLOYEES',
               max_read_label       => 'EMP:IS');
END;
```

例 7.58 和例 7.59 给没有标签的用户分配标签，对于已有标签的用户，可以采取添

加类别元素的方法给原有标签添加类别元素值，如下面的示例所示。

例 7.60　用户 SEC_MGR 执行以下操作，给 ALL_EXECS 用户群的标签添加 SALES、DEV 和 IS 三个类别值：

```
BEGIN
  sa_user_admin.add_compartments
                    (policy_name      => 'ESBD',
                     user_name        => 'ALL_EXECS',
                     comps            => 'SALES,DEV,IS');
END;
```

4．基于类别标签的访问控制演示

有了类别标签、有了基于类别标签的访问授权后，可以来看一看基于类别标签的访问控制情况。首先，把一些具有类别标签的信息添加到数据库表中，下面这个例子完成这项工作。

例 7.61　用户 SCOTT 执行以下操作，给数据库表 ANNOUNCEMENTS 插入 5 条具有类别标签的记录信息：

```
INSERT INTO scott.announcements
                    (MESSAGE, ROWLABEL)
          VALUES ('New updates to quotas have been assigned.',
                    char_to_label ('ESBD', 'MGR:SALES'));
INSERT INTO scott.announcements
                    (MESSAGE, ROWLABEL)
          VALUES ('New product release date meeting scheduled.',
                    char_to_label ('ESBD', 'MGR:DEV'));
INSERT INTO scott.announcements
                    (MESSAGE, ROWLABEL)
          VALUES ('Quota club trip destined for Hawaii.',
                    char_to_label ('ESBD', 'EMP:SALES'));
INSERT INTO scott.announcements
                    (MESSAGE, ROWLABEL)
          VALUES ('Source control software updates distributed next week.',
                    char_to_label ('ESBD', 'EMP:DEV'));
INSERT INTO scott.announcements
                    (MESSAGE, ROWLABEL)
          VALUES ('Firewall attacks increasing.',
                    char_to_label ('ESBD', 'EMP:IS'));
```

例 7.61 中的 CHAR_TO_LABEL 函数的作用是把字符串形式的标签转换为对应的标签标识号。

下面是演示基于类别标签的数据库记录读访问控制的一些例子。

例 7.62　用户 SCOTT 执行以下操作，把分配给 ALL_EMPLOYEES 用户群的标签映射为数据库会话标签，然后查询数据库表：

```
EXEC sa_session.set_access_profile ('ESBD','ALL_EMPLOYEES');
SELECT MESSAGE FROM scott.announcements;
```

查询结果是：

```
This message is to notify all employees...
```

例 7.63　用户 SCOTT 执行以下操作，把分配给 SALES_EMPLOYEES 用户群的标签映射为数据库会话标签，然后查询数据库表：

```
BEGIN
  sa_session.set_access_profile ('ESBD',
                                 'SALES_EMPLOYEES');
END;
SELECT MESSAGE
    FROM scott.announcements;
```

查询结果是：

```
This message is to notify all employees...
Quota club trip destined for Hawaii.
```

例 7.64　用户 SCOTT 执行以下操作，把分配给 DEV_MANAGERS 用户群的标签映射为数据库会话标签，然后查询数据库表，把信息和与其对应的标签显示出来：

```
BEGIN
  sa_session.set_access_profile ('ESBD',
                                 'DEV_MANAGERS');
END;
SELECT MESSAGE,
          label_to_char (ROWLABEL) "OLS Label"
    FROM scott.announcements;
```

查询结果是：

```
MESSAGE                                                          OLS Label
---------------------------------------------------------------- ----------
All Managers: employee compensation announcement... MGR
This message is to notify all employees...                         EMP
New product release date meeting scheduled.                    MGR:DEV
```

```
Source control software updates distributed next week.          EMP:DEV
```

例 7.64 中的 LABEL_TO_CHAR 函数的作用是把标签的标签标识号转换为对应的字符串形式的标签。

例 7.65　用户 SCOTT 执行以下操作，把分配给 ALL_EXECS 用户群的标签映射为数据库会话标签，然后查询数据库表，把信息和与其对应的标签显示出来：

```
BEGIN
  sa_session.set_access_profile ('ESBD',
                                 'ALL_EXECS');
END;
SELECT MESSAGE,
       label_to_char (ROWLABEL) "OLS Label"
  FROM scott.announcements;
```

查询结果是：

```
MESSAGE                                                    OLS Label
---------------------------------------------------------- ---------------
This message is only for the Executive Staff.              EXEC
All Managers: employee compensation announcement...        MGR
This message is to notify all employees...                 EMP
New updates to quotas have been assigned.                  MGR:SALES
New product release date meeting scheduled.                MGR:DEV
Quota club trip destined for Hawaii.                       EMP:SALES
Source control software updates distributed next week.     EMP:DEV
Firewall attacks increasing.                               EMP:IS
```

7.2.7　标签的组别元素的创建与应用

组别元素是具有层次关系的元素，即组别元素之间可以存在父子关系，父组别元素可以看到子组别元素中的记录，但兄弟组别元素之间的记录是互不可见的。

假设示例中的企业是一个跨国企业，对企业进行分组时，首先，整个企业可以看成是一个组（记为 CORP），如果按照地理区域进一步划分，可以分出美国组（记为 US）、纽约组（记为 NY）和洛杉矶组（记为 LA）等。也可以把欧洲、中东和非洲划为一个组（记为 EMEA），而亚太区为一个组（记为 APAC）。显然，CORP 是 US、EMEA 和 APAC 的父组，US 是 NY 和 LA 的父组。

1．组别元素的定义

以下的例子给出了组别元素的定义方法。

例 7.66　用户 SEC_MGR 执行以下操作，定义 CORP 组别元素：

```
BEGIN
  sa_components.CREATE_GROUP
                        (policy_name      => 'ESBD',
                         long_name        => 'Corporate',
                         short_name       => 'CORP',
                         group_num        => 1,
                         parent_name      => NULL);
END;
```

参数 PARENT_NAME 定义父组的名称，例 7.66 中的 NULL 值表示该组没有对应的父组。

例 7.67　用户 SEC_MGR 执行以下操作，定义 US、NY 和 LA 三个组别元素：

```
BEGIN
  sa_components.CREATE_GROUP
                        (policy_name      => 'ESBD',
                         long_name        => 'United States',
                         short_name       => 'US',
                         group_num        => 100,
                         parent_name      => 'CORP');
  sa_components.CREATE_GROUP
                       (policy_name       => 'ESBD',
                        long_name         => 'New York',
                        short_name        => 'NY',
                        group_num         => 110,
                        parent_name       => 'US');
  sa_components.CREATE_GROUP
                       (policy_name       => 'ESBD',
                        long_name         => 'Los Angeles',
                        short_name        => 'LA',
                        group_num         => 120,
                        parent_name       => 'US');
END;
```

例 7.68　用户 SEC_MGR 执行以下操作，定义 EMEA 和 APAC 两个组别元素：

```
BEGIN
  sa_components.CREATE_GROUP
                       (policy_name     => 'ESBD',
```

```
                    long_name      => 'Europe Middle_East Africa',
                    short_name     => 'EMEA',
                    group_num      => 200,
                    parent_name    => 'CORP');
  sa_components.CREATE_GROUP
                   (policy_name    => 'ESBD',
                    long_name      => 'Asia and Pacific',
                    short_name     => 'APAC',
                    group_num      => 300,
                    parent_name    => 'CORP');
END;
```

2．三元标签的创建

前面已经定义了若干等级元素、类别元素和组别元素，利用这些元素可以创建出三种元素齐全的标签。

例 7.69　用户 SEC_MGR 执行以下操作，创建与产品销售类员工相对应的四个标签：

```
BEGIN
  -- 欧洲、中东和非洲区域的销售经理
  sa_label_admin.create_label
                   (policy_name    => 'ESBD',
                    label_tag      => 300,
                    label_value    => 'MGR:SALES:EMEA');
  -- 美国的销售经理
  sa_label_admin.create_label
                   (policy_name    => 'ESBD',
                    label_tag      => 310,
                    label_value    => 'MGR:SALES:US');
  -- 纽约的产品销售员工
  sa_label_admin.create_label
                   (policy_name    => 'ESBD',
                    label_tag      => 320,
                    label_value    => 'EMP:SALES:NY');
  -- 洛杉矶的产品销售员工
  sa_label_admin.create_label
                   (policy_name    => 'ESBD',
                    label_tag      => 330,
```

```
                label_value     => 'EMP:SALES:LA');
END;
```

例 7.70　用户 SEC_MGR 执行以下操作，创建与产品开发类员工相对应的三个标签：

```
BEGIN
  -- 美国的产品开发员工
  sa_label_admin.create_label
                (policy_name    => 'ESBD',
                 label_tag      => 400,
                 label_value    => 'EMP:DEV:US');
  -- 亚太区的产品开发员工
  sa_label_admin.create_label
                (policy_name    => 'ESBD',
                 label_tag      => 410,
                 label_value    => 'EMP:DEV:APAC');
  -- 企业的产品开发员工
  sa_label_admin.create_label
                (policy_name    => 'ESBD',
                 label_tag      => 450,
                 label_value    => 'MGR:DEV:CORP');
END;
```

3．基于三元标签的访问授权

与类别标签的分配类似，给用户分配包含等级、类别和组别三种元素的标签时，可以直接给用户分配整个标签，也可以给用户已拥有的标签添加组别元素值。

下面的例子给用户已经拥有的标签添加组别元素值。

例 7.71　用户 SEC_MGR 执行以下操作，给用户群 ALL_EXECS 的标签添加组别元素值 CORP：

```
BEGIN
  sa_user_admin.add_groups
                (policy_name    => 'ESBD',
                 user_name      => 'ALL_EXECS',
                 groups         => 'CORP');
END;
```

例 7.72　用户 SEC_MGR 执行以下操作，给 US 和 EMEA 组的销售经理用户群分配标签：

```
BEGIN
  sa_user_admin.set_user_labels
                    (policy_name          => 'ESBD',
                     user_name            => 'US_SALES_MGR',
                     max_read_label       => 'MGR:SALES:US');
  sa_user_admin.set_user_labels
                    (policy_name          => 'ESBD',
                     user_name            => 'EMEA_SALES_MGR',
                     max_read_label       => 'MGR:SALES:EMEA');
END;
```

例 7.73　用户 SEC_MGR 执行以下操作，给 NY 组和 LA 组的销售员工用户群分配标签：

```
BEGIN
  sa_user_admin.set_user_labels
                    (policy_name          => 'ESBD',
                     user_name            => 'NY_SALES_REP',
                     max_read_label       => 'EMP:SALES:NY');
  sa_user_admin.set_user_labels
                    (policy_name          => 'ESBD',
                     user_name            => 'LA_SALES_REP',
                     max_read_label       => 'EMP:SALES:LA');
END;
```

例 7.74　用户 SEC_MGR 执行以下操作，给用户群 DEV_MANAGERS 的标签添加组别元素值 CORP：

```
BEGIN
  sa_user_admin.add_groups
                    (policy_name      => 'ESBD',
                     user_name        => 'DEV_MANAGERS',
                     GROUPS           => 'CORP');
END;
```

例 7.75　用户 SEC_MGR 执行以下操作，给 APAC 组和 US 组的产品开发员工用户群分配标签：

```
BEGIN
  sa_user_admin.set_user_labels
                    (policy_name          => 'ESBD',
                     user_name            => 'APAC_DEVELOPER',
```

```
                    max_read_label      => 'EMP:DEV:APAC');
  sa_user_admin.set_user_labels
                   (policy_name         => 'ESBD',
                    user_name           => 'US_DEVELOPER',
                    max_read_label      => 'EMP:DEV:US');
END;
```

7.2.8　借助会话标签给记录标签赋值

除了可以显式地通过 INSERT 或 UPDATE 操作给数据库表的记录分配标签值以外，也可以使系统自动地为记录分配标签值。方法很简单，就是在 OLS 安全策略中设置 LABEL_DEFAULT 选项，如下面的示例所示。

例 7.76　用户 SEC_MGR 执行以下操作，在安全策略中设置 LABEL_DEFAULT 选项，以便系统自动给记录分配标签值：

```
BEGIN
  sa_policy_admin.remove_table_policy
                  (policy_name      => 'ESBD',
                   schema_name      => 'SCOTT',
                   table_name       => 'ANNOUNCEMENTS');
  sa_policy_admin.apply_table_policy
                  (policy_name      => 'ESBD',
                   schema_name      => 'SCOTT',
                   table_name       => 'ANNOUNCEMENTS',
                   table_options    => 'LABEL_DEFAULT,READ_CONTROL');
END;
```

在安全策略中设置了 LABEL_DEFAULT 选项后，在生成新记录时，如果不指定标签值，那么系统将自动取数据库会话标签的值作为记录标签的值，并填写到记录的标签字段中。

下面的示例让系统自动给记录分配标签值。

例 7.77　用户 SEC_MGR 执行以下操作，首先把用户群 US_SALES_MGR 的标签映射为数据库会话标签，然后向数据库表插入一条新记录，让系统自动给记录分配标签值：

```
BEGIN
  sa_session.set_access_profile ('ESBD',
                                 'US_SALES_MGR');
END;
INSERT INTO scott.announcements
```

```
        (MESSAGE)
    VALUES ('Presidential outlook for economy may affect revenue.');
```

下面的示例表明，例 7.77 达到了让系统自动给记录分配标签值的目的。

例 7.78　用户 SEC_MGR 执行以下操作，查询例 7.77 中刚插入的新记录的信息：

```
SELECT MESSAGE,
        label_to_char (ROWLABEL) "OLS Label"
  FROM scott.announcements
 WHERE MESSAGE LIKE 'Pres%';
```

查询结果是：

```
MESSAGE                                                    OLS Label
---------------------------------------------------------------- ---------------------
Presidential outlook for economy may affect revenue.     MGR:SALES:US
```

7.2.9　再谈基于标签的授权

前面几节结合对等级、类别和组别等标签元素的介绍，介绍了基于标签的授权方法。但是，相对来说，前面的介绍还是比较简单的，基于标签的授权还有更丰富的内容。

在给用户分配标签时，可以确定三种类型的标签值：标签的可能取值、默认的会话标签值和默认的记录标签值，如表 7-1 所示。

表 7-1　给用户分配标签时涉及的标签值类型

	标签的可能取值	默认的会话标签值（DEFAULT）	默认的记录标签值（ROW）
等级	（max_level, min_level） 最大等级和最小等级	映射到会话标签中的等级	自动分配给记录的等级
类别	（WRITE） 可以进行写操作的类别	映射到会话标签中的类别	自动分配给记录的类别
组别	（WRITE） 可以进行写操作的组别	映射到会话标签中的组别	自动分配给记录的组别

其中，标签的可能取值指定了可以分配给用户的标签的取值范围，默认的会话标签值指定了把用户标签映射为会话标签时分配给会话标签的值，默认的记录标签值指定了在设置了 LABEL_DEFAULT 选项的情况下系统自动分配给记录的标签值。

1．等级元素值的设置

为了简化问题的描述，首先考虑只有等级元素的标签的设置方法。

例 7.79　用户 SEC_MGR 执行以下操作，给用户 SCOTT 分配标签的等级元素值：

```
BEGIN
  sa_user_admin.set_levels
                        (policy_name     => 'ESBD',
                         user_name       => 'SCOTT',
                         max_level       => 'MGR',
                         min_level       => 'EMP',
                         def_level       => 'MGR',
                         row_level       => 'EMP');
END;
```

经过例 7.79 的设置后，用户 SCOTT 的最大等级为 MGR，最小等级为 EMP，默认的会话等级为 MGR，默认的记录等级为 EMP。也就是说，用户 SCOTT 的等级元素可以取 EMP 至 MGR 之间的值，当用户登录时，他拥有的会话等级将是 MGR，而当他执行 INSERT 或 UPDATE 操作时，系统自动给记录分配的等级将是 EMP。

下面的示例可以显示用户 SCOTT 当前的会话等级和默认记录等级。

例 7.80　用户 SCOTT 执行以下操作，检查当前的会话标签和默认记录标签；

```
SELECT sa_session.read_label ('ESBD') "Select Label",
          sa_session.row_label ('ESBD') "Insert Label"
   FROM DUAL;
```

查询结果是：

```
Select Label       Insert Label
---------------- ----------------
MGR                EMP
```

例 7.81　用户 SCOTT 执行以下操作，插入一条新记录，然后查看新记录的标签值：

```
INSERT INTO announcements
       VALUES ('Scott will be out of the office.');
SELECT MESSAGE, label_to_char (ROWLABEL) "OLS Label"
   FROM announcements
  WHERE MESSAGE LIKE 'Scott%';
```

查询结果是：

```
MESSAGE                                    OLS Label
------------------------------------------ ---------------
Scott will be out of the office.             EMP
```

例 7.81 插入记录时没有指定标签值，查询结果表明系统已经自动把记录的标签值设置为 EMP。

OLS 允许用户利用存储过程 SA_SESSION.SET_LABEL 临时改变会话标签的值，如下面的示例所示。（临时性表现在：当用户重新登录时，会话标签仍然取默认的会话标签值。）

例 7.82　用户 SCOTT 执行以下操作，把会话标签改为 EMP：

```
EXEC sa_session.set_label('ESBD','EMP');
```

类似地，用户可以利用存储过程 SET_ROW_LABEL 改变系统自动给记录设置的标签值，如下面的示例所示。

例 7.83　用户 SCOTT 执行以下操作，重新把会话标签设置为 MGR，把自动的记录标签设置为 MGR，插入一条新记录，查看表中记录的标签值：

```
EXEC sa_session.set_label('ESBD','MGR');
EXEC sa_session.set_row_label('ESBD','MGR');
INSERT INTO announcements
        VALUES ('Scott's management meeting is cancelled.');
SELECT MESSAGE, label_to_char (ROWLABEL) "OLS Label"
    FROM announcements;
```

查询结果是：

```
MESSAGE                                                          OLS Label
---------------------------------------------------------------- ----------
Scott will be out of the office.                                 EMP
Scott's management meeting is cancelled.                         MGR
All Managers: employee compensation announcement...              MGR
This message is to notify all employees...                       EMP
```

2．组别与类别元素的依赖关系

表 7-1 列出了给用户分配标签时可以设定的各种标签元素的值的类型情况，接着介绍了相应等级元素值的设置方法。类别元素和组别元素的相应值的设置方法与等级元素相应值的设置方法是类似的。与标签的可能取值对应的类别或组别值确定的是用户可以访问的所有类别或组别的范围；与默认的会话标签值对应的类别或组别值确定的是在会话标签中的类别或组别的值；与默认的记录标签值对应的类别或组别值确定的是系统自动给记录设置的类别或组别的值。

在基于标签的访问控制中，如果记录的标签同时含有类别元素和组别元素，并且访问的类型是写操作，那么允许访问的条件是：用户对记录标签中的某个组别拥有写权限，并且用户对记录标签中的所有类别拥有读权限。

也就是说，记录标签中的组别至少有一个出现在用户标签的组别值中，并且用户拥有对该组别的写权限；同时，记录标签中的所有类别都出现在用户标签的类别值中，并

且用户对这些类别都拥有读权限。

值得一提的是：以上条件只要求用户对记录标签中的类别拥有读权限，而没有要求写权限。这是记录标签中含有组别值的情况。如果记录标签中没有组别值，那么是要求用户拥有对类别的写权限的。

要在基于标签的访问控制中实施写访问控制，需要在安全策略中设置 WRITE_CONTROL 选项。

例 7.84　用户 SEC_MGR 执行以下操作，启动 OLS 机制的读访问控制、写访问控制和记录标签自动赋值等功能：

```
DECLARE
  l_options    VARCHAR2 (50)
    := 'HIDE,LABEL_DEFAULT,WRITE_CONTROL,READ_CONTROL';
BEGIN
  sa_policy_admin.remove_table_policy
                        (policy_name        => 'ESBD',
                         schema_name        => 'SCOTT',
                         table_name         => 'ANNOUNCEMENTS');
  sa_policy_admin.apply_table_policy
                        (policy_name        => 'ESBD',
                         schema_name        => 'SCOTT',
                         table_name         => 'ANNOUNCEMENTS',
                         table_options      => l_options);
END;
```

为了说明组别元素与类别元素共同影响写访问控制判定结果的情况，先给用户分配含有类别元素和组别元素的标签。

例 7.85　用户 SEC_MGR 执行以下操作，给用户 SCOTT 的标签添加 DEV 类别，并授权 SCOTT 对 DEV 类别进行读和写的权限：

```
BEGIN
  sa_user_admin.add_compartments
                      (policy_name      => 'ESBD',
                       user_name        => 'SCOTT',
                       comps            => 'DEV',
                       access_mode      => sa_utl.read_write,
                       in_def           => 'Y',
                       in_row           => 'Y');
END;
```

例 7.85 同时把 DEV 类别添加到默认的会话标签值和默认的记录标签值中。

例 7.86　用户 SEC_MGR 执行以下操作，给用户 SCOTT 的标签添加 SALES 类别，并把对 SALES 类别的只读权限授权给 SCOTT：

```
BEGIN
  sa_user_admin.add_compartments
                    (policy_name      => 'ESBD',
                     user_name        => 'SCOTT',
                     comps            => 'SALES',
                     access_mode      => sa_utl.read_only,
                     in_def           => 'Y',
                     in_row           => 'N');
END;
```

例 7.86 把 SALES 类别添加到默认的会话标签值中，但没有把它添加到默认的记录标签值中。

例 7.87　用户 SEC_MGR 执行以下操作，给用户 SCOTT 的标签添加 US 类别，并授权 SCOTT 对 US 类别进行读和写的权限：

```
BEGIN
  sa_user_admin.add_groups
                    (policy_name      => 'ESBD',
                     user_name        => 'SCOTT',
                     GROUPS           => 'US',
                     access_mode      => sa_utl.read_write,
                     in_def           => 'Y',
                     in_row           => 'Y');
END;
```

同样，例 7.87 同时把 US 组别添加到默认的会话标签值和默认的记录标签值中。请注意：因为 US 组是 NY 组和 LA 组的父组，所以，用户对 US 组拥有读权限和写权限意味着他自然地对 NY 组和 LA 组都拥有读权限和写权限。下面的示例能反映这个效果。

例 7.88　用户 SCOTT 执行以下操作，查看自己目前拥有读权限的标签和拥有写权限的标签：

```
SELECT sa_session.read_label ('ESBD') "Read Label",
       sa_session.write_label ('ESBD') "Write Label"
  FROM DUAL;
```

查询结果是：

```
Read Label                                  Write Label
------------------------------------------- ----------------------
MGR:DEV,SALES:US,NY,LA          MGR:DEV:US,NY,LA
```

给用户 SCOTT 分配了含有类别和组别元素的标签，现在来看一下数据库表中记录的标签情况。

例 7.89　用户 SCOTT 执行以下操作，查看 ANNOUNCEMENTS 表中记录标签的情况：

```
SELECT label_to_char (ROWLABEL) "OLS Label",
        COUNT (ROWLABEL) "Total Records Labeled"
    FROM announcements
 GROUP BY ROWLABEL;
```

查询结果是：

```
OLS Label                          Total Records Labeled
---------------------------------- ----------------------------
MGR                                2
EMP                                2
MGR:SALES                          1
MGR:DEV                            1
EMP:SALES                          1
EMP:DEV                            1
MGR:SALES:US                       1
EMP:SALES:NY                       1
EMP:SALES:LA                       1
```

下面，通过 DELETE 操作来查看写访问控制的情况。

例 7.90　用户 SCOTT 执行以下操作，删除类别为 SALES 的记录，然后再查看表中记录标签的情况：

```
DELETE FROM announcements
        WHERE label_to_char (ROWLABEL) LIKE '%:SALES:%';
SELECT label_to_char (ROWLABEL) "OLS Label",
        COUNT (ROWLABEL) "Total Records Labeled"
    FROM announcements
 GROUP BY ROWLABEL;
```

查询结果是：

```
OLS Label                        Total Records Labeled
-------------------------------- --------------------------
```

MGR	2
EMP	2
MGR:SALES	1
MGR:DEV	1
EMP:SALES	1
EMP:DEV	1

从例 7.88 的查询结果可知，用户 SCOTT 对 SALES 类别只有读权限，而没有写权限，但他对 US、NY 和 LA 组拥有写权限，对比例 7.89 和例 7.90 的查询结果可知，类别为 SALES 且组别为 US、NY 或 LA 的记录均已删除，但类别为 SALES 且没有组别元素的记录没有删除。不难看出，被删除的记录的标签的等级值均不高于用户的标签的等级值，且均不低于用户标签等级的最小值。

7.3　本章小结

本章以 Oracle 实现的虚拟专用数据库机制（VPD 机制）和基于标签的安全机制（OLS 机制）为代表，介绍了增强数据库系统安全性的技术和方法。

VPD 机制是一种细粒度的、基于内容的记录级访问控制机制，它的基本思想是动态地、透明地给数据库访问语句附加上合适的谓词，限定用户所能访问到的数据记录。

VPD 机制是绑定到表、视图或同义词（Synonym）等数据库对象之上的访问控制机制，本章统一以表对象为代表进行讨论。

VPD 机制根据安全策略实施访问控制，每个安全策略由一个策略函数来实现，每个策略函数产生一个字符串类型的返回值，这个返回值构成一个谓词，一个策略函数可以与一个特定模式中的一张特定的表关联起来，每当要对这张表进行访问时，相应的谓词就自动附加到对应的访问语句上，从而在访问控制中发挥作用。

安全策略可以明确地针对 SELECT、INSERT、UPDATE、DELETE 或 INDEX 等具体类型的语句来实施。如果在登记一个安全策略时指定了访问语句的类型，那么该安全策略将只对指定类型的语句起作用。

VPD 机制支持对同一张表实施多个安全策略，当多个安全策略实施到同一张表上时，系统对多个安全策略的控制实行逻辑“与”操作。有时，两个安全策略可能会得到冲突的结果。可以对安全策略实行分组管理，根据具体的应用环境启用一些安全策略和禁用一些安全策略，以避免安全策略冲突情况的发生。

VPD 机制支持敏感字段驱动的访问控制功能，这种面向字段敏感性的功能对于支持私密性安全需求的实现具有重要意义，它允许把敏感信息和非敏感信息存放在一起，同时，能够确保敏感信息不会泄露给无关的用户。

OLS 机制的基本思想是引入一种称为标签（ Label）的安全属性，给主体和客体都

设定这种安全属性，当一个主体请求对一个客体进行访问时，把主体的标签与客体的标签进行对比，以对比的结果为依据判定是否允许进行相应的访问。

OLS 机制是一种记录级的数据库访问控制机制，就客体而言，它为数据库表中的记录分配标签；就主体而言，它为用户分配标签。记录的标签表示记录中的数据具有的敏感程度，用户的标签表示用户拥有的对记录进行访问的授权。

传统的标签由两类元素组成：一类元素称为等级；另一类元素称为类别。一个传统的标签就是由等级元素和类别元素构成的一个二元组。OLS 机制在传统标签思想的基础之上引入了分组的思想，用等级、类别和组别三类元素来定义标签，OLS 机制的一个标签就是由等级元素、类别元素和组别元素构成的一个三元组。

OLS 机制为应用提供了灵活的标签定义支持。OLS 规定标签中的等级元素必须取一个值，而且只能取一个值。但是，标签中的类别和组别元素可以取一个值，也可以取多个值，还可以取 0 个值（即不取值）。

运用 OLS 机制实施数据库访问控制，需要进行创建安全策略、定义标签元素、创建有效标签、实施安全策略和分配用户标签五项工作。创建安全策略就是创建一个安全策略框架，以便在该框架中描述标签信息、用户授权信息和待保护的数据库对象信息。定义标签元素就是给出构成标签的等级、类别和组别等元素的定义。创建有效标签就是利用已定义的标签元素创建将要使用的各种标签。实施安全策略就是把一个安全策略实施到一张数据库表中。分配用户标签就是把已创建的标签分配给某个实际的数据库用户或某个用户群。

给数据库表中的记录分配标签值可以通过 INSERT 或 UPDATE 操作显式地完成，也可以通过系统自动地完成。只要在安全策略中设置 LABEL_DEFAULT 选项，在生成新记录时不指定标签值，系统将会自动地为记录分配标签值。

给用户分配标签时，可以确定三种类型的标签值：标签的可能取值、默认的会话标签值和默认的记录标签值。标签的可能取值指定了可以分配给用户的标签的取值范围，默认的会话标签值指把用户标签映射为会话标签时分配给会话标签的值，默认的记录标签值指系统自动分配给记录的标签值。

在 OLS 机制中，一个用户能够对一条记录进行读操作，必须同时满足以下条件：

（1）用户会话标签中的等级必须大于或等于记录标签中的等级；

（2）如果记录标签中含有组别元素，那么用户会话标签中的组别必须至少包含记录标签中的一个组别，并且用户拥有对该组别的读访问授权；

（3）用户会话标签中的类别必须包含记录标签中的所有类别。

在 OLS 机制中，一个用户能够对一条记录进行写操作，必须同时满足以下条件：

（1）记录标签中的等级必须大于或等于用户的最小等级；

（2）记录标签中的等级必须小于或等于用户会话标签中的等级；

（3）如果记录标签中含有组别元素，那么用户会话标签中的组别必须至少包含记录

标签中的一个组别，并且用户拥有对该组别的写访问授权；

（4）用户会话标签中的类别必须包含记录标签中的所有类别；

（5）如果记录标签中的组别不取值，用户必须对记录标签中的所有类别拥有写访问授权。



习题 7

1. 虚拟专用数据库（VPD）机制是如何体现出它能够使公共数据库的用户感受到自己享有“专用”数据库的服务的？

2. 虚拟专用数据库（VPD）机制是通过动态地把特定的____（参数、公式、谓词）附加到数据库访问语句上来实现对数据库的访问控制的，附加的内容相当于_____（不等式、索引、WHERE 子句）。

3. 利用虚拟专用数据库机制进行访问控制，需要针对____（表、字段、记录）定义访问控制规则，访问控制____（可以、不能）对访问的类型加以区分。

4. 作为一个用户，为了运用虚拟专用数据库（VPD）机制实施数据库访问控制，需要完成哪些主要工作？试举例加以说明。

5. 虚拟专用数据库（VPD）机制实施的访问控制策略对数据库管理员（DBA）是否起作用？恶意用户能否通过设置 VPD 规则使 DBA 失去访问数据库的能力？为什么？

6. 虚拟专用数据库（VPD）机制是否允许特定用户不受 VPD 访问控制的制约？何以见得？

7. 虚拟专用数据库（VPD）机制根据敏感字段进行访问控制的基本思想是什么？这对实际应用有什么意义？简要说明其基本用法。

8. Oracle 基于标签的安全机制（OLS）根据安全标签实施对数据库的访问控制，需要分配标签的客体是____（表、字段、记录），需要分配标签的主体是____（用户、进程、系统）。

9. Oracle 基于标签的安全机制（OLS）所定义的安全标签与贝尔−拉普杜拉模型定义的安全标签有什么共同之处和不同之处。

10. 作为一个用户，为了运用 Oracle 基于标签的安全机制（OLS）实施数据库访问控制，需要完成哪些主要工作？

11. 在 Oracle 基于标签的安全机制（OLS）的控制下，主体 S 想要从数据库中读客体 O 的信息，试简要描述 OLS 判断是否允许 S 读 O 的基本过程，并举例加以说明。

12. 在 Oracle 基于标签的安全机制（OLS）的控制下，主体 S 想要向数据库中写客体 O 的信息，试简要描述 OLS 判断是否允许 S 写 O 的基本过程，并举例加以说明。

13. 简要说明在 Oracle 基于标签的安全机制（OLS）中给用户分配的安全标签的以下取值的含义，并说明它们的用途：标签的可能取值、默认的会话标签值、默认的记录

标签值。

14. 在 Oracle 基于标签的安全机制（OLS）中给用户分配安全标签与在贝尔–拉普杜拉模型中给用户分配安全标签的方法有什么不同？

15. 在 Oracle 基于标签的安全机制（OLS）中，设 ca、cb、cc 是三个类别，ga、gb、gc 是三个组别，而且，ga 是 gb 和 gc 的父组，用户 U 和记录 1～10 的安全标签如下：

用户 U 的安全标签：lu = (200:ca,cb:ga)

记录 1 的安全标签：lr1 = (200)

记录 2 的安全标签：lr2 = (100)

记录 3 的安全标签：lr3 = (200:ca)

记录 4 的安全标签：lr4 = (200:cb)

记录 5 的安全标签：lr5 = (200:cc)

记录 6 的安全标签：lr6 = (100:ca)

记录 7 的安全标签：lr7 = (100:cb)

记录 8 的安全标签：lr8 = (200:cb:ga)

记录 9 的安全标签：lr9 = (200:cb:gb)

记录 10 的安全标签：lr10 = (200:cb:gc)

用户 U 的安全等级取值范围是 100～200，他对 ca 拥有读权限和写权限，对 cb 拥有读权限，对 ga 拥有读权限和写权限，请问用户 U 对哪些记录拥有读权限？对哪些记录拥有写权限？

第 8 章
系统完整性保护



完整性（Integrity）是信息安全的经典要素之一，在信息安全和可信计算中具有重要的意义。本章介绍系统完整性的度量与保护的基本思想、技术和方法。主要内容包括完整性的经典模型、基于系统安全引导的完整性、基于安全协处理器的完整性、基于安全中央处理器的完整性、内核主导的完整性度量和文件系统的完整性检查等。

8.1　完整性的经典模型

和机密性（Confidentiality）一样，完整性（Integrity）是信息安全的经典要素之一。信息的机密性指信息不被非法获取的特性，信息的完整性指信息不被非法修改的特性。非法修改包含两层意思：一是在没有得到授权的情况下对信息进行修改；二是虽有授权但不按授权的方式对信息进行修改。

例 8.1　系统 A 外的恶意用户 M 未曾拥有访问系统 A 的授权，更不拥有修改系统 A 中的程序 P 的授权，但他设法攻入了系统 A，并在程序 P 中注入了一段恶意代码，这是未获授权而修改信息的情形。

例 8.2　系统 A 中的文件 F 记录的是大学新生录取名单，系统授权用户 U 修改文件 F，“张三”不应出现在文件 F 中，但用户 U 把“张三”的名字添加到了文件 F 中，这是违反授权方式修改信息的情形。

例 8.1 和例 8.2 所描述的用户行为都属于非法修改信息的行为，都破坏了信息的完整性。

完整性保护的任务一方面是通过访问控制来防止完整性遭受破坏，另一方面是通过完整性度量来判断完整性是否遭到破坏。完整性访问控制模型为前一项任务提供支持，完整性度量模型为后一项任务提供支持。

针对例 8.1 的案例，系统的完整性访问控制机制要能够防止用户 M 修改程序 P，系统的完整性度量机制要能够判断程序 P 是否已被修改。

下面介绍两个典型的完整性访问控制模型和一个典型的完整性度量模型。鉴于前面几章对访问控制问题进行了较多的介绍，本章后面的内容将主要从完整性度量的角度讨论系统完整性保护问题。

8.1.1　毕巴模型

毕巴（K.J. Biba）于 1977 年提出了一个典型的完整性访问控制模型，人们把它称为毕巴模型（Biba 模型）。

毕巴模型用完整性级别来对完整性进行量化描述。完整性级别是有序的，可以进行比较。

定义 8.1　对于任意两个完整性级别 i_1 和 i_2，用“＜”表示完整性绝对支配关系，用以下式子

$$i_1 < i_2$$

表示完整性级别 i_2 支配完整性级别 i_1，其意义是完整性级别为 i_2 的实体比完整性级别为 i_1 的实体具有更高的完整性。这里所说的实体既可以是主体也可以是客体。

通常，实体 e 的完整性级别可以表示为 $i(e)$。在完整性绝对支配关系的基础上，可以进一步定义完整性普通支配关系。

定义 8.2　对于任意两个完整性级别 i_1 和 i_2，用“≤”表示完整性普通支配关系，用以下式子

$$i_1 \leqslant i_2$$

表示完整性级别 i_2 要么支配完整性级别 i_1，要么与完整性级别 i_1 相同。

自然地，完整性级别相同的实体具有相同的完整性。

完整性级别与可信度有密切的关系，完整性级别越高意味着可信度越高。如果程序 P_1 的完整性级别比程序 P_2 的高，则我们对程序 P_1 正确工作的信心比对程序 P_2 的更足。如果数据 D_1 的完整性比数据 D_2 的高，则我们相信数据 D_1 比数据 D_2 更准确、更可靠。如果进程 P 的完整性级别比客体 O 的高，则我们认为进程 P 比客体 O 更可信。

毕巴模型针对主体的“读”、“写”和“执行”操作进行完整性访问控制，可以用 $\underline{r}$、$\underline{w}$ 和 $\underline{x}$ 分别表示这三类操作，这些操作可以用约定 8.1 的方式进行描述。

约定 8.1　用 $S\ \underline{r}\ O$ 表示主体 S 可以读客体 O，用 $S\ \underline{w}\ O$ 表示主体 S 可以写客体 O，用 $S_1\ \underline{x}\ S_2$ 表示主体 S_1 可以执行（即启动）主体 S_2。

毕巴模型定义了信息传递路径的概念，用于刻画访问控制策略。

定义 8.3　一个信息传递路径是一个客体序列 $O_1, O_2, \cdots, O_{n+1}$ 和一个对应的主体序列 $S_1, S_2, \cdots, S_n$，其中，对于所有的 i（$1 \leqslant i \leqslant n$），有 $S_i\ \underline{r}\ O_i$ 和 $S_i\ \underline{w}\ O_{i+1}$。

信息传递路径的定义的意思是：对于客体序列中任意一个客体 O_i（$1 \leqslant i \leqslant n$），有一个主体 S_i 和它对应，主体 S_i 能读客体 O_i 中的信息，并能把信息写到客体 O_{i+1} 中。因此，客体序列确定了一个信息传递路径，信息可以沿着客体序列，从序列左端的客体传递到序列右端的客体。显然，通过主体序列中的主体依次进行的“读”和“写”操作，客体 O_1 中的信息可以传递到客体 O_{n+1} 中。

对于“写”和“执行”操作，毕巴模型定义了以下规则。

规则 8.1　“写”操作的实施由下面的规则（1）控制，“执行”操作的实施由下面的规则（2）控制：

（1）当且仅当 $i(O) \leqslant i(S)$ 时，主体 S 可以写客体 O。

（2）当且仅当 $i(S_2) \leqslant i(S_1)$ 时，主体 S_1 可以写主体 S_2。

规则 8.1 中的（1）规定高完整性级别的主体可以修改低完整性级别的客体，低完整性级别的主体不能修改高完整性级别的客体。规则 8.1 中的（2）规定高完整性级别的主体可以启动低完整性级别的主体，低完整性级别的主体不能启动高完整性级别的主体。

这是自然的，因为，如果低完整性级别的主体能够修改高完整性级别的客体，就意味着可信度低的主体能够修改可信度高的客体，那么，这很有可能破坏客体的完整性。如果低完整性级别的主体能够启动高完整性级别的主体，由于高完整性级别的主体能够修改高完整性级别的客体，那么，这等同于间接允许低完整性级别的主体修改高完整性级别的客体，这也很有可能破坏客体的完整性。

例 8.3　设 $PROC_1$ 和 $PROC_2$ 是两个进程，PROG 是一个系统程序，进程 $PROC_1$ 能把给定代码插入到给定程序中，进程与程序的完整性级别的关系是：

$$i(PROC_2) < i(PROC_1) = i(PROG)$$

在完整性控制的范围内，进程 $PROC_1$ 能够修改程序 PROG。如果低完整性级别的主体能够启动高完整性级别的主体，那么，就算进程 $PROC_2$ 是恶意进程，它可以启动进程 $PROC_1$，并指示进程 $PROC_1$ 把恶意代码 M_{code} 插入到程序 PROG 中，这样，便破坏了系统程序 PROG 的完整性。

对于“读”操作，通过定义不同的规则，毕巴模型可以呈现为三种略有不同的形式，相应的模型分别称为低水标（Low-Water-Mark）模型、环（Ring）模型和严格完整性（Strict Integrity）模型。

1．毕巴低水标模型

毕巴模型的低水标形式在规则 8.1 的基础上实施以下规则。

规则 8.2　设 $i_{after}=\min(i(S), i(O))$，如果主体 S 读客体 O，那么，“读”操作实施后，主体 S 的完整性级别被调整为 i_{after}，即“读”操作实施前的主体 S 的完整性级别和客体 O 的完整性级别这两个级别中较低的那个级别。

从实际效果上看，实施规则 8.2 时，如果 $i(S)$高于 $i(O)$，则 i_{after} 等于 $i(O)$，否则，i_{after} 等于 $i(S)$，即当主体读完整性级别比其自身低的客体时，主体的完整性级别降为客体的完整性级别，否则主体的完整性级别保持不变。

高完整性级别的主体读低完整性级别的客体，意味着该主体对低可信度的信息有一定的依赖性，我们对它的信任程度会因此而降低。把它的完整性级别降低到与客体的完整性级别相同，有助于防止低完整性级别的信息污染高完整性级别的主体。

规则 8.1 中的（1）旨在防止直接的非法修改行为的发生。直接的非法修改指的是主

体和客体的完整性级别标记能直接反映出其不合理性的修改。

规则 8.2 旨在防止间接的非法修改行为的发生。间接的非法修改指的是主体和客体原有的完整性级别标记虽反映不出其不合理性，但实际的完整性已预示着其不合理性的修改。例如，由于接受了低完整性的信息，主体的完整性实际上已经降低，但完整性级别标记还没反映出来。

所以，毕巴低水标模型立足于既防止直接的非法修改也防止间接的非法修改。

毕巴低水标模型对信息传递路径有约束作用，这可以通过以下定理来表达。

定理 8.1　在毕巴低水标模型的控制下，如果系统中存在一个从客体 O_1 到客体 O_{n+1} 的信息传递路径，那么，对于任意的 $n>1$，必有 $i(O_{n+1})\leqslant i(O_1)$。

定理 8.1 表明：在信息传递路径中，信息不会从低完整性级别的客体传向高完整性级别的客体。

2．毕巴环模型

毕巴模型的环形式不关心间接的非法修改问题，只关心直接的非法修改问题，它在规则 8.1 的基础上实施以下规则。

规则 8.3　不管完整性级别如何，任何主体都可以读任何客体。

实际上，环模型只根据规则 8.1 对“写”操作和“执行”操作进行控制，对“读”操作不实施任何控制。

3．毕巴严格完整性模型

毕巴模型的严格完整性形式根据主体和客体的完整性级别对“读”操作进行严格的控制，它在规则 8.1 的基础上实施以下规则。

规则 8.4　当且仅当 $i(S)\leqslant i(O)$ 时，主体 S 可以读客体 O。

把规则 8.4 和规则 8.1 中的（1）结合起来可知，在严格完整性模型中，当且仅当主体和客体拥有相同的完整性级别时，主体可以同时对客体进行“读”操作和“写”操作。

在规则 8.4 的控制下，如果主体 S 能够对客体 O 进行“读”操作，那么，客体 O 的完整性级别 $i(O)$必然支配主体 S 的完整性级别 $i(S)$。在这种情况下，如果实施规则 8.2，则主体 S 的完整性级别在“读”操作实施前后相同，即实施和不实施规则 8.2 的效果是相同的，或者说，不实施规则 8.2 已等同于实施了规则 8.2。也就是说，只要实施了规则 8.4，就隐含地实施了规则 8.2。

可见，实施了严格完整性模型的规则，就隐含地实施了低水标模型的规则，因而，和低水标模型一样，严格完整性模型同时防止对实体进行直接的或间接的非法修改，同时，在严格完整性模型的控制下，定理 8.1 的结论显然成立。

通常，提及毕巴模型时，一般都是指毕巴严格完整性模型。另外，毕巴模型的完整性级别并非只是单纯的基数型属性，它的定义可以包含基数型的完整性等级和集合型的完整性类别等元素。

8.1.2　克拉克-威尔逊模型

克拉克（D.D. Clark）和威尔逊（D.R. Wilson）于 1987 年提出了另一个典型的完整性访问控制模型，人们把它称为克拉克-威尔逊模型（Clark-Wilson 模型），简称克-威模型（C-W 模型）。

1．基本概念的定义

克-威模型是一个面向事务（Transaction）的模型，事务是操作的集合，一个事务由一组操作构成。事务作用在数据上，克-威模型的出发点是既要确保数据的完整性，又要确保事务的完整性。

例 8.4　在一个银行转账系统中，允许把资金从 A 账户转到 B 账户，从 A 账户转出一笔金额为 amount 的资金，需要执行以下操作：

$$A.balance = A.balance - amount \quad (1)$$

其中，A.balance 表示 A 账户的余额。把金额为 amount 的资金转入 B 账户，需要执行以下操作：

$$B.balance = B.balance + amount \quad (2)$$

一次转账必须执行（1）和（2）两个操作，否则银行总资金的余额就会不平衡。（1）和（2）这两个操作构成了一个转账事务 T_{trans}。

事务的执行可能使系统的状态发生变化。在例 8.4 中，事务 T_{trans} 的执行使 A 账户的余额减少，B 账户的余额增加，系统的状态发生了变化。

定义 8.4　如果一个事务使系统从一个有效的状态转换到另一个有效的状态，则这样的事务称为良构（Well-formed）事务。

在例 8.4 中，如果银行总资金的余额平衡时系统处于有效状态，那么，事务 T_{trans} 是一个良构事务。

克-威模型针对事务在数据上的作用建立完整性控制的体系框架，它把系统中的所有数据项划分为两大类：约束数据项和非约束数据项。

定义 8.5　在一个系统中，处于完整性模型的控制范围之内的数据项称为约束数据项（CDI，Constrained Data Item），处于完整性模型的控制范围之外的数据项称为非约束数据项（UDI，Unconstrained Data Item）。

克-威模型把所有 CDI 都满足完整性策略要求时的系统状态定义为有效状态。与状态相关，克-威模型定义了两类事务：完整性验证过程和转换过程。

定义 8.6　在克-威模型中，用于验证系统是否处于有效状态的事务称为完整性验证过程（IVP，Integrity Verification Procedure），用于使系统从一个有效状态转换到另一个有效状态的事务称为转换过程（TP，Transformation Procedure）。

在例 8.4 的案例中，验证银行总资金余额是否平衡的事务是 IVP，实现转账功能的事务是 TP。克-威模型要实现的 TP 是良构事务。

根据有效状态的定义，在有效状态下，CDI 的完整性是有保障的，要确保 CDI 的完整性就是要确保系统处于有效状态。

克-威模型实现完整性控制的基本思想是：假设系统在某个初始时刻处于有效状态，在系统运行过程中，模型确保只有 TP 能够对 CDI 进行操作，进而确保只有 TP 能够改变系统的状态。

由于克-威模型要实现的 TP 是良构事务，所以系统在运行过程中总是处于有效状态，因而，系统的完整性可以得到保障。

2．模型的规则

系统可以实现 TP，也可以确保只有 TP 能够对 CDI 进行操作，但系统本身无法确保 TP 一定是良构的。只能根据特定的完整性策略通过系统以外的途径去证明 TP 是良构的。

克-威模型采取两类措施实现系统的完整性：一类是由系统实施的措施；另一类是证明系统实施的措施的有效性的措施。与此相对应，克-威模型定义两类规则：实施规则和证明规则。

定义 8.7　在克-威模型中，由系统实施的规则称为实施（Enforcement）规则，简记为 E 规则；用于证明系统实施的规则的有效性的规则称为证明（Certification）规则，简记为 C 规则。

以下三个规则构成了克-威模型的基础框架，它确保 CDI 的内部一致性。

规则 C1　IVP 必须验证所有的 CDI 在 IVP 运行时刻都处于有效状态中。

规则 C2　必须验证所有的 TP 都是有效的，即它们必须把处于一个有效状态中的 CDI 转换到另一个有效状态。安全管理员必须为每一个 TP 及其所能操作的 CDI 集合定义对应关系，对应关系的格式是 $(TP_i, (CDI_a, CDI_b, CDI_c, \cdots))$，这个关系一一列出了给定的 TP 可以操作的所有 CDI。

规则 E1　系统必须维护规则 C2 中描述的关系表，对于任意的 CDI，系统必须确定只有关系表中列出的对应 TP 才能对该 CDI 进行操作。

以下规则对克-威模型的基础框架进行拓展，以支持职责分离原则。

规则 E2　系统必须维护一个确定用户、TP 和 CDI 集合之间关系的关系表，关系的格式为 (UserID, TP_i, (CDI_a, CDI_b, CDI_c, …))，其中，UserID 是用户标识。系统必须确保只有某个关系中指定的用户才能执行指定的 TP 对指定的 CDI 进行操作。

规则 C3　必须证明规则 E2 中的关系表满足职责分离要求。

以下规则实现用户身份认证，以支持规则 E2 中要用到的用户标识。

规则 E3　系统必须认证试图执行 TP 的每个用户的身份。

几乎所有实施完整性的系统都要求对所有 TP 的执行进行审计，提供审计记录。审计记录是特定形式的 CDI，以下规则提供审计支持。

规则 C4　必须证明所有的 TP 都向一个只能以附加方式写的 CDI（日志）写入足够的信息，以便能够重现 TP 的操作过程。

系统中除了包含 CDI 以外，也包含 UDI，从系统外部进入到系统中的新信息就属于 UDI，如用户从输盘上输入的新信息。以下规则说明对 UDI 的处理方法。

规则 C5　必须证明接收 UDI 输入的 TP 对 UDI 的任何可能取值，要么只进行有效转换，要么不进行转换。即 TP 把输入的 UDI 转换为 CDI 或拒绝相应的 UDI 输入。在典型情况下，这样的 TP 是编辑程序。

为了确保克-威模型的有效性，不能忽略任何一个证明规则。例如，用户不能创建并运行一个没有经过证明的新的 TP，否则系统就达不到它的有效性目的。为此，系统需要以下规则加以约束。

规则 E4　只有有权对实体（如 TP）进行证明的主体才能修改这些实体与其他实体（如 CDI）之间的关系表。有权对某个实体（如 TP 或 CDI）进行证明的主体不能拥有与该实体相关的执行权。

修改实体的关系表就是修改实体的访问控制配置属性，假设某实体的属主有权修改实体的关系表，那么，根据规则 E4，该实体的属主不能拥有与该实体相关的执行权，但是，实际上，一个实体的属主是拥有与该实体相关的执行权的，这样便产生了矛盾，所以，假设不能成立，即一个实体的属主不能修改该实体的访问控制配置属性，因此，规则 E4 确定了克-威模型不是一个自主访问控制模型，它是一个强制访问控制模型。

例 8.5　设系统的关系表中包含关系 (U_1, TP_i, (CDI_a, CDI_b, CDI_c, …))，请问，能够

由用户 U_1 证明 TP_i 的有效性或证明 CDI_a 的完整性吗？

解答：如果用户 U_1 有权证明 TP_i 的有效性，根据规则 E4，该用户就无权执行 TP_i，但已知条件说明用户 U_1 能够执行 TP_i，这是矛盾的，所以，不能由用户 U_1 证明 TP_i 的有效性。

同样，如果用户 U_1 有权证明 CDI_a 的完整性，根据规则 E4，该用户就无权在 CDI_a 上执行操作，但已知条件说明用户 U_1 可以执行 TP_i 在 CDI_a 上进行操作，这也是矛盾的，所以，不能由用户 U_1 证明 CDI_a 的完整性。

例 8.5 说明了规则 E4 的实际含义。

3．模型的概括

克-威模型定义了 C1～C5 五个证明规则和 E1～E4 四个实施规则，这九个规则共同定义一个实施一致的完整性策略的系统。由这九个规则构成的克-威模型可以用图 8.1 进行概括。

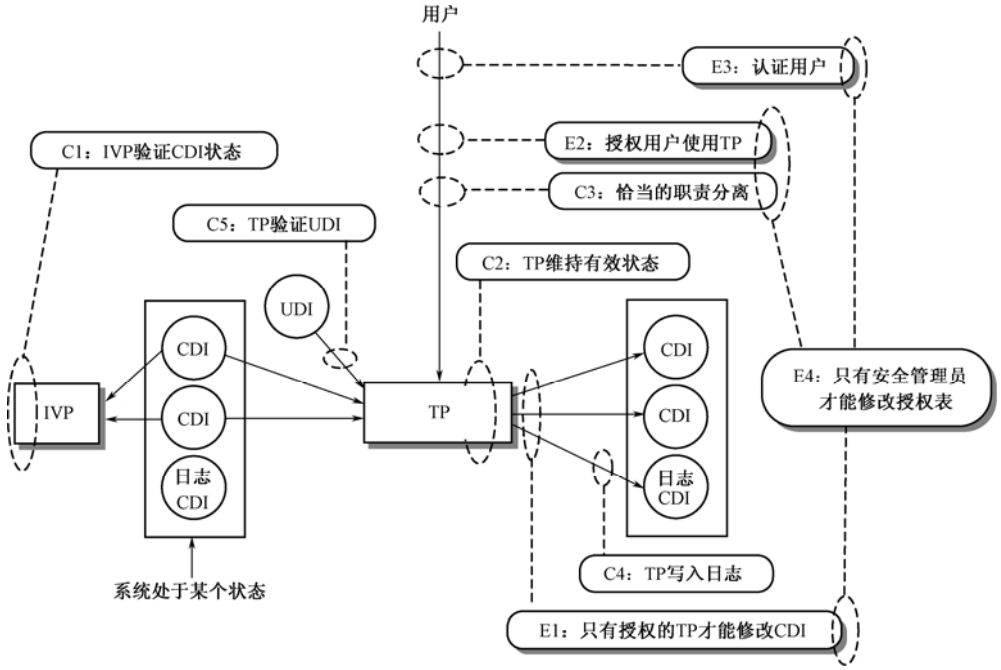


图 8.1　克-威模型的系统完整性规则

克-威模型从良构事务和职责分离两个方面构造完整性控制框架，它把系统中的数据项划分为约束数据项和非约束数据项两大类，通过转换过程和完整性验证过程两类过程定义系统的行为，依靠证明规则和实施规则两种规则实现对系统行为的约束，进而支持一致的完整性控制策略。

8.1.3　莫科尔树模型

莫科尔（R.C. Merkle）于 1979 年提出了一个典型的完整性度量模型，即哈希树（Hash Tree）模型，人们把它称为莫科尔树（Merkle Tree）模型。

1．哈希函数

数据完整性度量的常用方法之一是使用哈希函数。哈希函数是一种单向函数，即根据自变量可以很容易地计算出函数的值，但根据函数的值不可能逆向推出自变量。例如，对于函数的以下一般形式

$$y = h(x)$$

如果 h 是单向函数，那么，根据 h 和 x 可以很容易地计算出 y，但根据 h 和 y 不可能计算出 x。

对于给定的哈希函数，不管自变量的取值如何，函数值的长度是固定的，而且，不同的自变量对应不同的函数值。

定理 8.2　设 h 是哈希函数，且有

$$y_1 = h(x_1)$$
$$y_2 = h(x_2)$$

那么，y_1 和 y_2 的长度一定相等，而且，若 x_1 不等于 x_2，则 y_1 不等于 y_2。

采用哈希函数度量数据的完整性的基本方法是：在不同的时刻采用同一个哈希函数计算同一个数据的哈希值，假设已知在某个时刻数据是完整的，把其他时刻的哈希值与该时刻的哈希值进行比较，如果相等，则可断定数据的完整性在其他时刻没有被破坏，否则可断定数据的完整性在其他时刻已被破坏。

例 8.6　假设数据 D 在初始状态下是完整的，试说明采用哈希函数度量数据 D 在任意时刻的完整性的方法。

解答：设 h 为哈希函数，t_0 为初始时刻，t_x 为一个任意时刻。在 t_0 时刻，计算

$$y_{t_0} = h(D)$$

在 t_x 时刻，计算

$$y_{t_x} = h(D)$$

因为已知数据 D 在初始状态下是完整的，所以，y_{t_0} 对应的是数据 D 的完整性有保障的状态。对比 y_{t_0} 和 y_{t_x}，如果 y_{t_x} 等于 y_{t_0}，则可断定数据 D 在 t_x 时刻的完整性是有保障的，否则可断定数据 D 在 t_x 时刻的完整性已受到破坏。

如果存在两个不同的自变量，使某哈希函数计算出相同的函数值。例如，对于以上的哈希函数 h，存在 x_1 不等于 x_2 但 y_1 等于 y_2 的情况，那么，就说该哈希函数（如 h）发生了碰撞。

哈希函数根据哈希算法进行设计，MD5 和 SHA-1 等是常用的典型哈希算法。采用哈希函数进行完整性度量的基础是使用没有碰撞的哈希算法。例如，例 8.6 中的哈希函数 h 应该是不会发生碰撞的。

2．哈希树

定义 8.8　哈希树是用于计算数据项的哈希值的二叉树，树中的每个叶节点与一个数据项对应，每个内部节点与该节点的两个子节点的值的连接结果的哈希值对应。

哈希树是由莫科尔提出来的，因而也称为莫科尔树。设 h 是一个哈希函数，n 是哈希树中一个内部节点的值，c_1 和 c_2 是该节点的两个子节点的值，则有

$$n = h(c_1 \| c_2)$$

其中，“||”表示连接操作。例如，如果 c_1 的值为“Integrity”，c_2 的值为“Measurement”，则 $c_1 \| c_2$ 的值为“IntegrityMeasurement”。

例 8.7　设 h 是一个哈希函数，D_1、D_2、D_3 和 D_4 是 4 个数据项，则与这 4 个数据项对应的哈希树可以用图 8.2 表示，并且，以下等式成立：

$$n_1 = h(D_1 \| D_2); \quad n_2 = h(D_3 \| D_4); \quad n_0 = h(n_1 \| n_2)$$

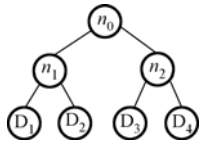


图 8.2　一棵哈希树

莫科尔树模型是在数据项分割的基础上实现数据项的完整性验证的模型，它的基本出发点是依靠较少的内存空间开销实现较快的数据完整性验证。设 D 是一个给定的数据项，可以根据需要把它分割成 D_1、D_2、…、D_n 等 n 个数据项，即

$$D = D_1 \| D_2 \| \cdots \| D_n$$

莫科尔树模型要解决的问题是设计一个算法，使对于任意一个数据项 D_i $(1 \leqslant i \leqslant n)$，该算法能够快速验证数据项 D_i 的完整性，并且占用较少的内存空间。

为了实现对任意一个数据项 D_i 的完整性的验证，莫科尔树模型按照以下方式定义一个递归函数 f。

定义 8.9　设 h 是一个哈希函数，D 是给定的数据项，i 和 j 是自然数$(1 \leqslant i, j \leqslant n)$，函数 $f(i, j, D)$的值由以下方法确定：

（1）$f(i,i,D) = h(D_i)$；

（2）$f(i,j,D) = h(\, f(i, (i+j-1)/2, D) \,\|\, f((i+j+1)/2, j, D))$。

$f(i,j,D)$是 D_i、D_{i+1}、…、D_j 的函数，可用于验证 D_i、D_{i+1}、…、D_j 的完整性。特别地，$f(1,n,D)$可用于验证 D_1、D_2、…、D_n 的完整性。

在一棵具有 n 个叶节点的哈希树中，$f(i,i,D)$ $(1 \leqslant i \leqslant n)$对应叶节点的哈希值，$f(1,n,D)$对应树根节点的哈希值。

运用定义 8.9 中定义的递归函数 f，可以验证任意一个数据项 D_i 的完整性，验证的方法假设树根的哈希值 $f(1,n,D)$是已知且正确的，验证的过程沿着从树根到树叶的方向依次展开。

例 8.8　假设数据项 D 被分割成 D_1、D_2、…、D_8 等 8 个数据项，已知对应哈希树中的哈希函数为 h，试给出运用定义 8.9 中定义的递归函数 f 验证数据项 D_5 的完整性的过程。

解答：哈希树的叶节点数 n 为 8，假设 $f(1,8,D)$是已知且正确的，按照以下步骤验证数据项 D_5 的完整性：

（1）设法获取 $f(1,4,D)$和 $f(5,8,D)$，验证以下等式是否成立，如果成立，则可断定 $f(1,4,D)$和 $f(5,8,D)$是正确的：

$$f(1,8,D) = h(f(1,4,D) \| f(5,8,D))$$

（2）第（1）步已获得 $f(5,8,D)$的值并证明它是正确的，设法获取 $f(5,6,D)$和 $f(7,8,D)$，验证以下等式是否成立，如果成立，则可断定 $f(5,6,D)$和 $f(7,8,D)$是正确的：

$$f(5,8,D) = h(f(5,6,D) \| f(7,8,D))$$

（3）第（2）步已获得 $f(5,6,D)$的值并证明它是正确的，设法获取 $f(5,5,D)$和 $f(6,6,D)$，验证以下等式是否成立，如果成立，则可断定 $f(5,5,D)$和 $f(6,6,D)$是正确的：

$$f(5,6,D) = h(f(5,5,D) \| f(6,6,D))$$

（4）第（3）步已获得 $f(5,5,D)$的值并证明它是正确的，对于给定的数据项 D_5，验证以下等式是否成立，如果成立，则可断定数据项 D_5 是完整的：

$$f(5,5,D) = h(D_5)$$

如果以上第（1）～（4）各步都能顺利完成，则证明数据项 D_5 的完整性没有问题，否则证明数据项 D_5 的完整性已被破坏。

例 8.8 的完整性验证过程可以通过图 8.3 加以更加形象地描述，图 8.3 中带箭头的虚线标出了从哈希树的根节点到叶节点的路径。

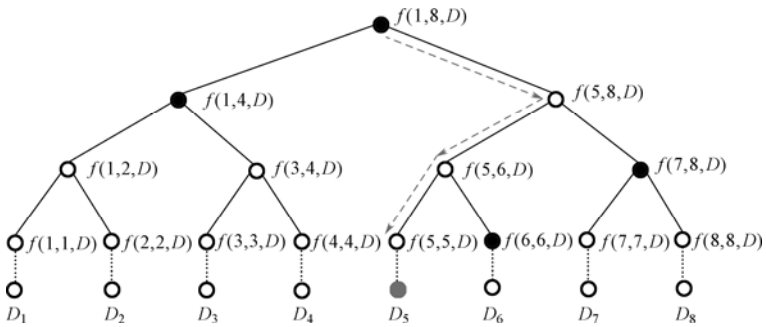


图 8.3　数据项 D_5 的完整性验证路径

在例 8.8 给出的完整性验证方法中，为了验证数据项 D_5 的完整性，需要用到的数

值有

$$f(1,8,D)、f(1,4,D)、f(5,8,D)、f(5,6,D)、f(7,8,D)、f(5,5,D)、f(6,6,D)、\cdots\cdots \quad (1)$$

但是，由于：

$f(5,5,D)$可以由 D_5 计算得到；

$f(5,6,D)$可以由 $f(5,5,D)$和 $f(6,6,D)$计算得到；

$f(5,8,D)$可以由 $f(5,6,D)$和 $f(7,8,D)$计算得到。

所以，实际需要用到的数值有

$$f(1,8,D)、f(1,4,D)、f(7,8,D)、f(6,6,D)、\cdots\cdots \quad (2)$$

（2）中的数值对应着图 8.3 中的黑色实心节点，它们构成了数据项 D_5 的完整性验证路径。完整性验证路径的一般定义如下。

定义 8.10　从哈希树的根节点 $f(1,n,D)$到叶节点 $f(i,i,D)$存在一条唯一的路径 P，由与 P 上的节点相邻的 P 以外的节点的哈希值（及 $f(1,n,D)$）组成的集合称为数据项 D_i 的完整性验证路径。这里，两节点相邻的意思是指它们之间有一条边直接相连。

定理 8.3　只要拥有数据项 D_i 的完整性验证路径中的正确数值，就能验证数据项 D_i 的完整性。

例如，例 8.8 中的数据项 D_5 的完整性验证路径是（2），如果（2）中的数值是已知的且正确的，那么，根据该组数值便可验证数据项 D_5 的完整性。

验证的方法是：根据 D_5 计算出 $f(5,5,D)$，根据 $f(5,5,D)$和 $f(6,6,D)$计算出 $f(5,6,D)$，根据 $f(5,6,D)$和 $f(7,8,D)$计算出 $f(5,8,D)$，根据 $f(1,4,D)$和 $f(5,8,D)$计算出 $f(1,8,D)$，如果计算出的 $f(1,8,D)$与已知的 $f(1,8,D)$相等，则 D_5 是完整的，否则 D_5 是不完整的。

8.2　基于系统安全引导的完整性

系统引导（Boot 或 Bootstrap）指的是从计算机上电到操作系统进入正常工作状态的过程，系统安全引导的目的是使系统能够顺利地引导起来，并确保引导起来的操作系统的完整性是有保障的。

阿玻（W.A. Arbaugh）、凡柏（D.J. Farber）和史密斯（J.M. Smith）于 1997 年提出了一个系统安全引导模型，称为 AEGIS 引导模型。本节以该模型为基础讨论通过系统安全引导来建立系统的完整性的方法。

8.2.1　系统引导的一般过程

在讨论系统的安全引导之前，有必要较深入地认识一下计算机系统从通电到操作系统开始正常工作的这个时段里发生了哪些事情。不失一般性，可以结合与 IBM 个人计

算机兼容的普通计算机进行考察。

抽象是计算机学科中常用的分析问题的基本方法，可以采用该方法把计算机系统的引导过程划分为若干个抽象层，使每个抽象层对应系统引导的一个阶段，最低的抽象层对应引导的最早阶段，最高的抽象层对应引导的最后阶段。

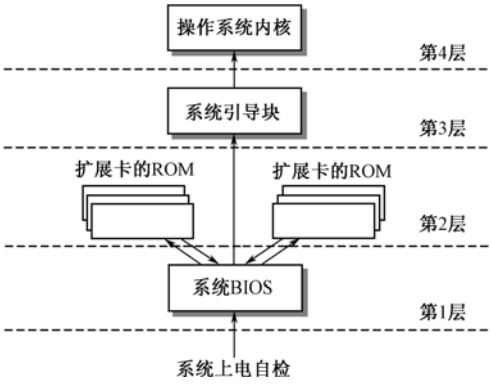


图 8.4　普通 PC 的一般引导过程

普通 PC 的一般引导过程如图 8.4 所示，可以把普通计算机系统引导的一般过程划分为 4 个抽象层，它们分别表示系统引导的 4 个阶段。

给计算机通电是启动计算机的引导过程的最直接的方法。从断电状态进入通电状态时，计算机的硬件结构自动启动系统的上电自检（POST，Power On Self Test）过程，促使 CPU 执行位于处理器复位向量指示的入口点处的指令。

POST 过程的启动也就是系统引导过程的开端。POST 操作检测硬件组件的基本状态，包括 CPU 的状态。除了初始的 CPU 自检之外，POST 操作的检测工作在系统 BIOS 的控制下进行。POST 操作在引导过程的第 1 层进行。

执行完 POST 操作后，系统 BIOS 寻找系统中可能存在的扩展卡，如音频卡、视频卡等。如果找到有效的扩展卡，系统 BIOS 把控制权交给相应的扩展卡中的 ROM，扩展卡的 ROM 代码在引导过程的第 2 层执行，执行完毕，将控制权返回给系统 BIOS。

扩展卡的检查及相应的 ROM 代码执行完成后，系统 BIOS 调用初始引导代码，该初始引导代码属于系统 BIOS 的一部分，它根据 CMOS 中的定义查找可引导设备（如光盘、硬盘等），找到可引导设备后，从可引导设备中把系统引导块装入到内存中，并把控制权交给内存中的引导块代码。

系统引导块代码在引导过程的第 3 层执行，它负责把操作系统内核装入到内存中。如果引导块由多个部分组成，则引导初始代码装入的是主引导程序，主引导程序再装入次引导程序，次引导程序再装入次次引导程序……如此依次进行下去，直到装入所有的引导程序。

最后装入的引导程序把操作系统内核装入内存，并把控制权交给它。操作系统内核在引导过程的第 4 层执行。操作系统进入正常工作状态后，系统引导过程结束。

在理想情况下，系统引导过程应该以线性顺序执行，即按各个抽象层由低到高依次地顺序执行，每一层代码完成执行后，把控制权交给与它相邻的高一层的代码，直到操作系统内核进入正常工作状态。但是，图 8.4 表明，系统引导过程有可能呈现“星状”结构，而并非纯线性结构，因为，图 8.4 中第 2 层的代码完成执行时，把控制权返回给

第 1 层的代码，而不是如预期的那样把控制权交给第 3 层的代码。

8.2.2　系统的可信引导过程

根据 8.2.1 节的介绍，在系统引导的一般过程中，组件 A 把控制权交给组件 B 时，它并不了解组件 B 的完整性状况，即使组件 B 的完整性已经受到破坏，组件 A 依然把控制权交给组件 B，组件 B 依然能够运行，所以，在系统引导过程结束时，没办法知道投入运行的操作系统的完整性是否已经受到破坏，这样的引导过程不是可信的引导过程。

定义 8.11　可信引导是这样的一种引导过程：它确保引导过程中获得控制权的所有组件的完整性都是没有受到过破坏的，进而确保引导起来的操作系统的完整性是有保障的。

为了实现可信引导的目标，需要对组件的完整性进行验证。组件的哈希值可以作为组件的指纹，用于进行组件的完整性验证。验证的方法是对比组件的原始指纹和即时指纹，两者相同时，组件的完整性良好，否则组件的完整性受损。

例 8.9　设在系统的可信引导过程中，涉及组件 A 把控制权交给组件 B 的操作，试说明控制权的可信交接方法。

解答：设系统中保存有组件 B 的原始指纹 h_{B0}，组件 A 把控制权交给组件 B 前，计算组件 B 的指纹 h_{Bt}，并对比 h_{Bt} 和 h_{B0}，如果 h_{Bt} 等于 h_{B0}，则把控制权交给组件 B，否则不把控制权交给组件 B。

通过指纹的对比，可以验证组件的完整性。采取这样的完整性验证方法，一方面需要保存组件的原始指纹，另一方面需要进行即时指纹的计算和对比，即进行完整性验证，这可由专门设计的代码来完成。保存的原始指纹的完整性及进行完整性验证的代码的完整性也必须得到保障。

AEGIS 模型在系统中增设了一个专用的 ROM 卡，称为 AEGIS ROM，用于存储组件的原始指纹。同时，该模型把系统 BIOS 划分成两部分，分别称为主 BIOS 和辅 BIOS，主 BIOS 中包含执行完整性验证任务的代码，辅 BIOS 包含 BIOS 的其他成分及 CMOS。

AEGIS 模型假设 AEGIS ROM 和主 BIOS 是可信的，即它们的完整性由模型外的其他措施来提供保障，在模型中，它们包含可信软件，是值得信赖的完整性验证的根。

在以上设计思想的指导下，图 8.4 所示的引导过程可以扩展为如图 8.5 所示的形式，用于支持系统的可信引导。其中，作为可信根，AEGIS ROM 和主 BIOS 位于系统可信引导过程的第 0 层，这一层组件的完整性被默认为是良好的。

图 8.5 中的辅 BIOS 代替了图 8.4 中的系统 BIOS，位于引导过程的第 1 层。系统的引导过程到第 4 层的操作系统结束，图 8.5 中增加了由用户程序构成的第 5 层，它表示可信引导确保的完整性可以潜在地拓展到应用程序中。

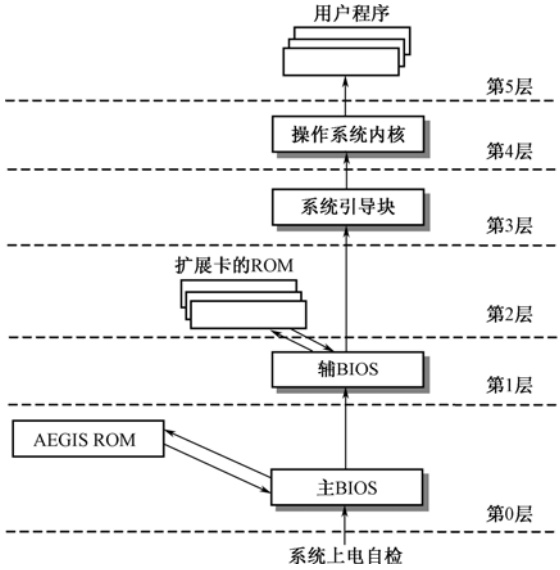


图 8.5　系统的可信引导过程

系统的可信引导从第 0 层逐级向第 4 层推进。模型假设第 0 层的组件是可信的，所以，无须进行完整性验证，只是，为了避免出现 ROM 失效，主 BIOS 验证其自身的地址空间的校验和（Checksum）。

把控制权交给辅 BIOS 之前，第 0 层的主 BIOS 计算第 1 层的辅 BIOS 当时的指纹，并与辅 BIOS 的原始指纹进行对比，从而验证辅 BIOS 的完整性，如果完整性良好，则把控制权交给辅 BIOS，第 1 层的辅 BIOS 开始执行。

第 1 层的辅 BIOS 验证第 2 层的扩展卡的 ROM 的完整性，如果完整性良好，则把控制权交给扩展卡的 ROM，第 2 层的扩展卡的 ROM 开始执行。

类似地，第 1 层的辅 BIOS 中的初始引导代码验证第 3 层的主引导块的完整性，如果完整性良好，则把控制权交给系统的主引导块，第 3 层的主引导块开始执行。

如果系统有次引导块，则主引导块验证次引导块的完整性，完整性良好时，把控制权交给次引导块。第 3 层的最后一个引导块验证第 4 层的操作系统内核的完整性，如果完整性良好，则把控制权交给操作系统内核，第 4 层的操作系统内核开始运行。

在以上各步的完整性验证过程中，如果发现完整性受损，可以中止系统的引导过程，这样，当操作系统能够进入正常的工作状态时，就可以断定它的完整性是良好的。因此，可信引导能够确保投入运行的操作系统的完整性。

8.2.3　组件完整性验证技术

为简单起见，8.2.2 节介绍的方法通过保存组件的原始指纹来支持组件的完整性验证。采用公开密钥密码体系的数字签名技术对原始指纹进行签名，在系统中保存原始指

纹的数字签名，而不是直接保存原始指纹，可以增强原始指纹的抗篡改性和真实性。

例 8.10　设系统保存组件原始指纹的数字签名以支持可信引导，试说明组件 A 验证组件 B 的完整性的方法。

解答：设组件 B 的原始指纹的数字签名为$\{h_{B0}\}K_{PRV\text{-}S}$，系统的公钥证书为$\{K_{PUB\text{-}S}\}K_{PRV\text{-}CA}$，证书发放机构的公钥为$K_{PUB\text{-}CA}$。

根据公钥$K_{PUB\text{-}CA}$和证书$\{K_{PUB\text{-}S}\}K_{PRV\text{-}CA}$可以得到系统的公钥$K_{PUB\text{-}S}$，根据公钥$K_{PUB\text{-}S}$和数字签名$\{h_{B0}\}K_{PRV\text{-}S}$可以得到组件 B 的原始指纹$h_{B0}$。

根据组件 B 可以计算出组件 B 的指纹h_{Bt}，对比h_{Bt}和h_{B0}，如果h_{Bt}等于h_{B0}，则组件 A 可以断定组件 B 的完整性是良好的，否则组件 A 断定组件 B 的完整性已被破坏。

AEGIS 模型采用数字签名来实现组件的完整性验证，它在 AEGIS ROM 中存储公钥证书和组件原始指纹的数字签名。

可信引导过程中的完整性验证操作的集合构成一个完整性验证链，可信引导借助完整性验证链来维护系统的完整性。如图 8.5 所示的可信引导过程的完整性验证链可描述为

$$I_0 = \text{True}, \qquad \text{验证链的根}$$

$$I_{i+1} = \begin{cases} I_i \wedge V_i(L_{i+1}) & \text{对于 } i = 0, 3, 4 \\ I_i \wedge \sum_{l=1}^{n} V_i(L_{i+1}^{l}) & \text{对于 } i = 1 \\ I_i \wedge V_{i-1}(L_{i+1}) & \text{对于 } i = 2 \end{cases}$$

I_i是个布尔值，表示第 i 层的完整性，I_i取值为真（True）表示第 i 层的完整性是良好的，I_i取值为假（False）表示第 i 层的完整性已被破坏。“∧”是逻辑“与”操作。V_i是第 i 层使用的完整性验证函数，V_i接收的参数表示待验证的抽象层，它的返回结果是个布尔值。验证函数计算给定抽象层组件的指纹（哈希值），并结合存储在系统中的组件原始指纹的数字签名验证组件的完整性。n 表示扩展卡的个数。这里的“Σ”表示多个值的逻辑“与”操作。

8.2.4　AEGIS模型的安全引导过程

可信引导的主要目标是确保顺利引导起来的操作系统内核的完整性是良好的，但它不确保操作系统内核一定能够顺利引导起来。在可信引导的过程中，一旦发现组件的完整性受损，可以中止系统的引导过程，在这样的策略的指导下，可信引导的最终结果有两个：一是完整性良好的操作系统内核顺利地引导起来；二是操作系统内核没能顺利地引导起来。也就是说，通过可信引导，顺利投入运行的操作系统内核一定是可信的，不可信的操作系统内核一定不可能顺利投入运行。

采取以上策略的可信引导不能抵御拒绝服务攻击，因为，出现组件完整性受损时，引导过程就会中止，所以，攻击者只要破坏引导过程中涉及的某个组件的完整性，就能

达到拒绝服务攻击的目的。

系统安全引导可以克服系统可信引导中存在的容易遭受拒绝服务攻击的弱点。

定义 8.12 安全引导是这样的一种引导过程：它不但要确保顺利引导起来的操作系统内核一定是完整性良好的，还要确保操作系统内核一定能够顺利引导起来。

为了实现系统安全引导的目标，需要在系统可信引导的基础上，增加系统恢复的功能。也就是说，在引导过程中，发现某个组件的完整性已受破坏时，不是中止引导过程，而是首先用验证过的组件副本对该组件进行恢复，然后使引导过程继续进行下去。

例 8.11 设在系统安全引导过程中，组件 A 的完整性已通过验证，试说明组件 A 把控制权交给组件 B 的方法。

解答：组件 A 计算组件 B 的指纹 h_{Bt}，结合组件 B 的数字签名$\{h_{\mathrm{B0}}\}K_{\mathrm{PRV\text{-}S}}$和系统的公钥证书$\{K_{\mathrm{PUB\text{-}S}}\}K_{\mathrm{PRV\text{-}CA}}$，验证组件 B 的完整性。

如果完整性验证通过，则组件 A 可把控制权交给组件 B。如果完整性验证没通过，则组件 A 首先请求系统恢复组件 B，然后再重新进行组件完整性的验证工作。

恢复组件 B 的方法是：首先获取验证过的组件 B 的副本 B_{verified}，然后把组件副本 B_{verified} 复制到组件 B 的地址空间中，取代组件 B。

由例 8.11 可知，为了实现系统恢复功能，一要保存验证过的组件副本，二要利用组件副本恢复完整性受损的组件。

AEGIS 模型是一个系统安全引导模型，它提供出现完整性受损时的系统恢复支持。增加系统恢复功能后，如图 8.5 所示的过程可以扩展为如图 8.6 所示的 AEGIS 模型的安全引导过程。

在 AEGIS 模型中，验证过的组件副本存储在 AEGIS ROM 中和网络主机中，其中，AEGIS ROM 中只存储 BIOS 的组件副本，其他组件副本存储在网络主机中。网络主机像用户程序那样处于引导过程的第 5 层，它的完整性及它与被引导系统间的通信的安全性和可靠性由 AEGIS 模型以外的其他措施来保障。

BIOS 组件的恢复工作相对比较简单，主要是把验证过的组件副本从 AEGIS ROM 的地址空间复制到 BIOS 组件的地址空间。执行这类恢复任务的代码存储在主 BIOS 中。

其他组件的恢复工作相对比较复杂，包括从网络主机中获取验证过的组件副本，并把它复制到完整性受损组件的地址空间中。这类工作由执行系统恢复任务的系统恢复专用内核来完成，系统恢复专用内核存储在 AEGIS ROM 中。

可见，AEGIS 模型的完整性验证和系统恢复支持主要是由第 0 层提供的，该层由 AEGIS ROM 和主 BIOS 构成，AEGIS ROM 包含公钥证书、数字签名、验证过的 BIOS 组件副本、系统恢复专用内核等。主 BIOS 包含执行完整性验证任务的代码和执行 BIOS 组件恢复任务的代码。

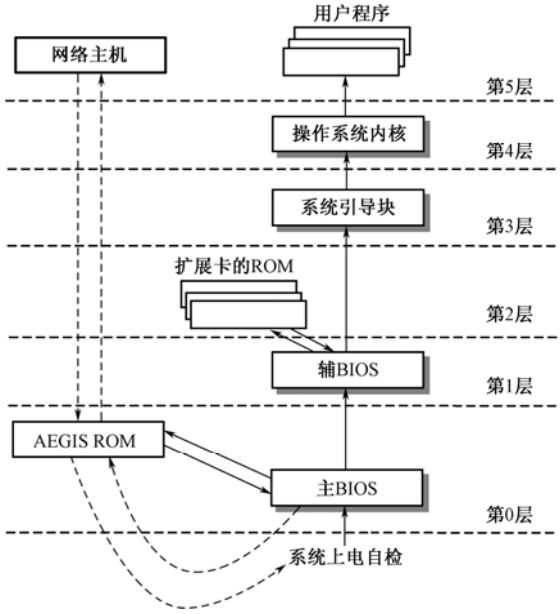


图 8.6　AEGIS 模型的安全引导过程

AEGIS 模型的安全引导过程与 8.2.2 节介绍的可信引导过程基本相同，不同之处是增加了系统恢复处理工作。

在 AEGIS 模型的安全引导过程中，发现 BIOS 组件完整性受损时，由存储在主 BIOS 中的执行 BIOS 组件恢复任务的代码，根据存储在 AEGIS ROM 中的验证过的 BIOS 组件副本，恢复完整性受损的组件。

发现 BIOS 组件以外的其他组件的完整性受损时，系统中止进行中的常规引导过程，转而引导存储在 AEGIS ROM 中的系统恢复专用内核，该专用内核进入运行后，根据完整性受损的组件，从网络主机中获取所需的验证过的组件副本，并恢复完整性受损的组件，最后，重新启动计算机系统。

计算机系统重新启动后，安全引导过程重新开始执行，此时，上次完整性受损的那个组件已被恢复，再次验证时，其完整性应该是良好的，引导过程可以向前推进一步。

所以，通过综合运用完整性验证和系统恢复的措施，安全引导过程可以确保系统一定能够顺利地引导起来，而且，可以确保顺利地引导起来的系统的完整性一定是良好的。

8.3　基于安全协处理器的完整性

在传统意义上，协处理器通常是完成特定任务的、独立的、从属于主处理器的处理单元，浮点运算协处理器就是这种类型的用于提高数学浮点运算速度的处理单元。

这里所说的安全协处理器与传统意义上的协处理器有所不同，它指的是具有独立物理封装结构的、内含独立的处理器的硬件组件，它可以插接在计算机的主板上，与中央

处理器并行地工作，为系统安全提供支持。

IBM 4758 是一款典型的安全协处理器。一个 4758 安全协处理器就是一个缩微的片上计算机，其内部拥有独立的操作系统，可以运行独立的应用软件。

从应用的角度看，安全性要求高的软件系统可以部署到插接在主板上的 4758 安全协处理器的内部，与部署在主机系统上的软件系统协同工作，提供联合的安全解决方案。

本节以 IBM 4758 为例，介绍安全协处理器的体系结构和工作原理，同时，考察安全协处理器实现系统完整性支持的方法。

8.3.1　IBM 4758 安全协处理器的硬件组织结构

IBM 4758 安全协处理器是封装在特殊物理外壳中的密封的缩微计算机，它唯一的对外接口是一个 PCI 总线接口，借助该接口，它可以插接到主机的 PCI 总线上，与主机进行通信。4758 拥有处理器、存储器、控制器、时钟等主要组件，它的硬件组织结构如图 8.7 所示。

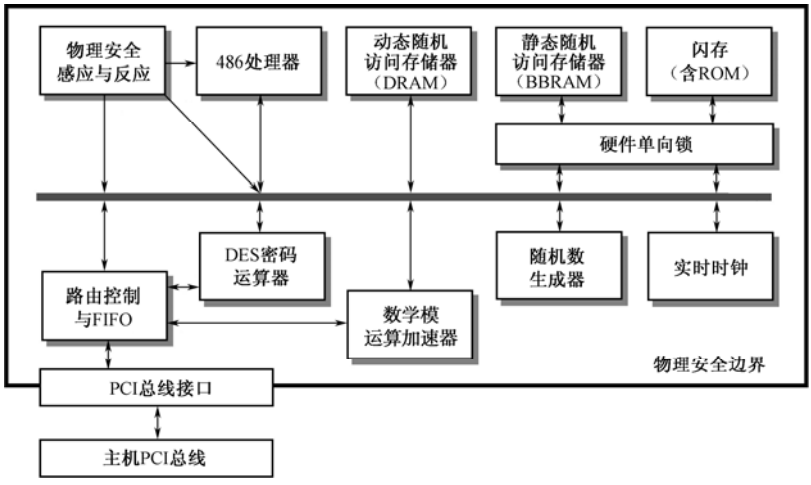


图 8.7　IBM 4758 安全协处理器的硬件组织结构

IBM 4758 的中央处理器是一个 486 档次的处理器，其他具有处理功能的组件包括数学模运算加速器、DES 密码运算器、随机数生成器等。

IBM 4758 的存储器包括易失性的动态随机访问存储器（DRAM）、非易失性电池支持的静态随机访问存储器（BBRAM）、闪存等类型。其中，DRAM 相当于普通计算机中的内存，程序装入其中后运行；BBRAM 用于保存密钥等秘密信息；闪存的一小部分由硬件线路设定为只读存储器（ROM），闪存的其他部分充当普通计算机中的硬盘。

IBM 4758 的硬件单向锁是一种状态不可逆的锁机制，它以硬件的手段实现对 BBRAM 和闪存空间的访问控制。该硬件锁的状态的表现形式是非负整数，由微型控制器维护。IBM 4758 上电时，硬件锁的状态被置为初始值 0，此后，锁的状态值只能增

加，不能减少，除非 IBM 4758 被复位，此时，锁的状态值被重置为 0。

IBM 4758 中的 BBRAM 可划分为可上锁的 BBRAM（LBBRAM，Lockable BBRAM）和不可上锁的 BBRAM 两种类型，前者用于保存固件的秘密信息，后者用于保存软件的秘密信息。可上锁的 BBRAM 可以划分成不同的区域，以便与不同的硬件锁状态相对应，从而根据区域实施访问控制。硬件锁对静态存储器区域的访问控制如表 8-1 所示。

表 8-1　硬件锁对静态存储器区域的访问控制

	锁状态 = 0	锁状态 = 1	锁状态 = 2	锁状态 = 3	锁状态 = 4
LBBRAM 区域 0	读，写	禁止访问	禁止访问	禁止访问	禁止访问
LBBRAM 区域 1	读，写	读，写	禁止访问	禁止访问	禁止访问
LBBRAM 区域 2	读，写	读，写	读，写	禁止访问	禁止访问
LBBRAM 区域 3	读，写	读，写	读，写	读，写	禁止访问

BBRAM：由备用电池供电支持的静态随机访问存储器（Battery-Backed RAM）。

由表 8-1 可以看出，硬件锁的状态为 0 时，各个 LBBRAM 区域都允许实施读访问和写访问；锁状态为 3 时，只有区域 3 允许实施读访问和写访问；锁状态为 4 时，所有区域均已禁止实施访问。

例 8.12　设 P_A 和 P_B 是 IBM 4758 安全协处理器中的两个程序，程序 P_B 在程序 P_A 之后运行，试说明利用硬件锁防止程序 P_B 获取程序 P_A 的密钥信息的方法。

解答：设程序 P_A 在硬件锁的 0 状态运行，把程序 P_A 的密钥信息 K_A 存放在 LBBRAM 的 0 号区域，这时，程序 P_A 能够访问密钥信息 K_A。

程序 P_A 完成任务后，先促使硬件锁的状态值加 1，再启动程序 P_B。程序 P_B 运行时，硬件锁的状态是 1（0+1=1），此时，硬件禁止任何程序访问 LBBRAM 的 0 号区域，又已知硬件锁的状态是不可逆的，所以，程序 P_B 不可能访问 LBBRAM 的 0 号区域，因而，它不可能获取程序 P_A 的密钥信息 K_A。

与 LBBRAM 的访问控制方法类似，可以把要保护的闪存空间划分成不同的区域，利用硬件锁对这些区域进行访问控制。表 8-2 描述了硬件锁在指定的闪存区域上施加访问控制的方法。

表 8-2　硬件锁对闪存区域的访问控制

	锁状态 = 0	锁状态 = 1	锁状态 = 2	锁状态 = 3	锁状态 = 4
闪存区域 1	读，写，清除	读，写，清除	读	读	读
闪存区域 2	读，写，清除	读，写，清除	读	读	读
闪存区域 3	读，写，清除	读，写，清除	读	读	读

根据表 8-2，在状态 0 和状态 1 下，硬件锁允许对受保护的各个闪存区域进行读、写和清除操作，在其他状态，只允许对受保护的各个闪存区域进行读操作。

由表 8-1 和表 8-2 可知，硬件锁施加访问控制的措施是使闪存的受保护区域成为“只读”区域，而使 LBBRAM 的受保护区域成为完全不可访问区域。不管是 LBBRAM 还是闪存，受保护区域都是在硬件中定义的。

IBM 4758 中的物理安全感应与反应单元给该安全协处理器装备了物理上的篡改反应功能，如果攻击者试图对该安全协处理器进行物理攻击，如试图撬开它的外壳，取出其中的存储器，以获取其中存储的信息，该安全协处理器能够感应出攻击的企图，并清除其中存储的所有信息。

8.3.2　IBM 4758 安全协处理器的软件层次结构

IBM 4758 对软件进行分层管理。软件的分层就是把可能出现在 IBM 4758 中的所有软件划分成相互独立的部分，并根据各部分软件运行的先后次序分配层次编号，先运行的软件被定义为低层次的软件，获得较小的层次编号，最低层称为第 0 层。IBM 4758 安全协处理器的软件层次结构如图 8.8 所示。

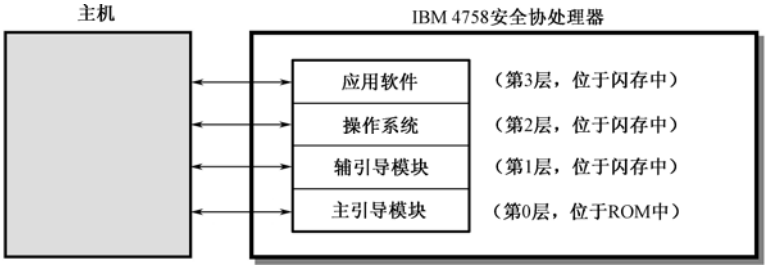


图 8.8　IBM 4758 安全协处理器的软件层次结构

在 IBM 4758 的软件层次结构中，程序代码被划分为 4 个层次，第 0 层是主引导模块，第 1 层是辅引导模块，第 2 层是操作系统，第 3 层是应用软件。其中，第 0 层的代码存放在 ROM 中，其他层的代码存放在闪存中。

第 0 层的代码是在 IBM 4758 出厂前安装到 ROM 中的，IBM 4758 模型把它定义为固件。其他层的代码是在 IBM 4758 出厂后才安装的，IBM 4758 模型把它们定义为软件。

在软件分层的基础上，IBM 4758 模型定义了“责任人”的概念，软件层次结构中的每一层都有相应的责任人和它对应，责任人肩负其所在层次的软件的管理和维护的职责。IBM 4758 提供了“建立责任人”的命令，由第 L 层的责任人执行该命令设定第（L+1）层的责任人。

图 8.9 给出了 IBM 4758 的软件层次结构中各个层次上的责任人的一个示例。第 0 层的责任人是主引导模块的责任人，也就是 IBM 4758 中的固件的责任人，由 IBM 4758 的生产厂家（即 IBM 公司）充当，在 4758 出厂前设定。其他层的责任人由低层的责任人执行“建立责任人”的命令进行设定。

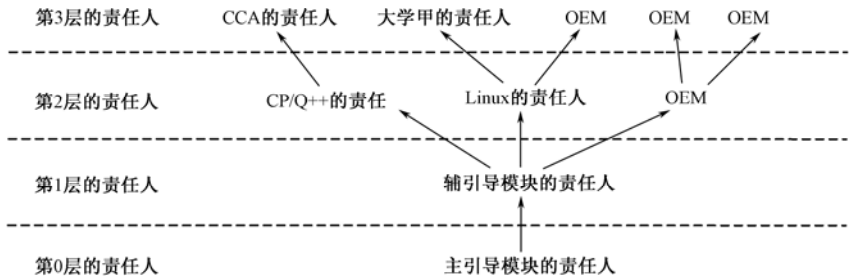


图 8.9　软件层次结构中对应的责任人

第 1 层的责任人是辅引导模块的责任人。第 2 层的责任人是操作系统的责任人，在不同的 IBM 4758 上可以安装不同的操作系统，如 CP/Q++操作系统、Linux 操作系统、其他 OEM 的操作系统等，每一种操作系统都有其相应的责任人，其中，CP/Q++是以市场销售的 CP/Q 操作系统为基础，由 IBM 为 4758 改造出的操作系统。

第 3 层的责任人是应用软件的责任人，同样，在不同的 IBM 4758 上可以安装不同的应用软件，每一种应用软件都有其相应的责任人。比起操作系统，应用软件的种类更加繁多，如 IBM 的公共密码体系结构（CCA）应用软件、大学甲的应用软件、其他 OEM 的应用软件等。

8.3.3　可信代码的安装与更新

代码责任人的重要作用之一是在代码的安装或更新过程中发挥代码认证的作用，从而确保安装或更新的代码是可信的。

这里，代码的安装指的是在把 IBM 4758 看做是裸机的情况下，把代码安装到其中；而代码的更新指的是相应的、可信的代码已经在 IBM 4758 中正常工作的情况下，更新代码的内容或有关信息。也就是说，可以建立下面的约定。

约定 8.2　对于任意的 L（$L \geqslant 1$），在安装第 L 层（$L \geqslant 1$）代码前，第 L 层代码不能正常工作；在更新第 L 层代码前，第 L 层代码能够正常工作。

约定 8.3　在考虑代码的安装或更新时，第 0 层代码总是正常工作的，而且是可信的。

IBM 4758 的第 0 层代码（即存储在 ROM 中的主引导模块代码）提供基于共享密钥算法的认证功能和安装第 1 层代码（即存储在闪存中的辅引导模块代码）的功能；第 1 层代码提供基于公开密钥算法的认证功能和安装、更新代码的功能。

1．代码的可信安装

在安装代码的过程中，我们采取先安装低层代码再安装高层代码的基本策略，为此，我们可以建立下面的约定。

约定 8.4　对于任意的 L（$L \geqslant 1$），在安装第 L 层（$L \geqslant 1$）代码前，第（$L-1$）层代码能够正常工作。

首先，考虑第 1 层代码（辅引导代码）的安装问题。此时，只有第 0 层的代码（主引导代码）在工作，只能使用共享密钥算法在 IBM 4758 与第 0 层的责任人之间进行相互认证。

IBM 4758 拥有共享密钥 $K(0)$，第 0 层的责任人知道该密钥，所以，他能够利用该密钥证实自己的身份，并向主引导代码签发“安装第 1 层代码”的命令，主引导代码验证该命令的有效性，如果命令有效，则执行该命令，把可信的辅引导代码安装到 IBM 4758 中。

辅引导代码正常工作后，就能够以它为基础，安装其他层的代码，并采用基于公开密钥算法的认证功能，确保代码的真实性，基本思路是由下一层的责任人认证本层的责任人的公钥的真实性，由本层的责任人认证要安装的代码的真实性。安装方法如下。

方法 8.1　对于任意的 L（$2 \leqslant L \leqslant 3$），按照以下步骤安装第 L 层的代码：

（1）由第（$L-1$）层的责任人为第 L 层的责任人签发公钥证书：

$$\{K_{\mathrm{PUB}}(L)\}K_{\mathrm{PRV}}(L-1) \qquad ①$$

其中，$K_{\mathrm{PUB}}(L)$ 表示第 L 层的责任人的公钥，$K_{\mathrm{PRV}}(L-1)$ 表示第（$L-1$）层的责任人的私钥。

（2）由第 L 层的责任人向 IBM 4758 签发“安装第 L 层代码”的命令：

$$\{C_{\mathrm{install}}(L,\ \mathrm{CODE}(L),\ K_{\mathrm{PUB\text{-}new}}(L),\ \cdots)\}K_{\mathrm{PRV}}(L) \qquad ②$$

其中，

$$C_{\mathrm{install}}(L,\ \mathrm{CODE}(L),\ K_{\mathrm{PUB\text{-}new}}(L),\ \cdots) \qquad ③$$

表示“安装第 L 层代码”的命令，$\mathrm{CODE}(L)$ 表示第 L 层代码，$K_{\mathrm{PUB\text{-}new}}(L)$ 表示第 L 层的责任人在代码安装后拟采用的新公钥，它可以与原来的公钥 $K_{\mathrm{PUB}}(L)$相同。

（3）安装第 L 层的代码时，第（$L-1$）层的代码已经顺利安装，所以，辅引导代码可以获得第（$L-1$）层的责任人的公钥 $K_{\mathrm{PUB}}(L-1)$，据此可以验证①式中的签名，从而获得第 L 层的责任人的公钥 $K_{\mathrm{PUB}}(L)$，据此又可以验证②式中的签名，从而获得如③式所示的“安装第 L 层代码”的命令。

（4）由辅引导代码根据③式所示的命令安装第 L 层代码。

代码真实性的认证思想可以描述为如图 8.10 的形式。即，在代码安装的过程中，每一层代码的真实性需要在下一层的责任人的帮助下进行认证，这由图 8.10 中的带箭头的实线表示。

IBM 4758 在把安装的代码保存到闪存中的同时，也把该代码的责任人的公钥保存到了闪存中，这一点，方法 8.1 的③式有所揭示，因此，有下面的推论。

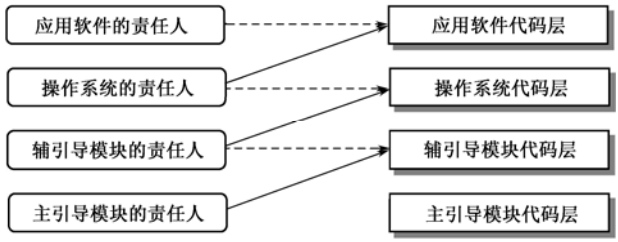


图 8.10　代码安装或更新的验证

推论 8.1　如果第 L 层（$L \geqslant 1$）的代码 CODE(L)已经顺利安装到 IBM 4758 中，那么，IBM 4758 可以从它的闪存中获得代码 CODE(L)的责任人的公钥 $K_{\text{PUB}}(L)$。

实际上，方法 8.1 的第（3）步已经利用了推论 8.1 的结果。

方法 8.1 的③式还揭示另一层意思：在安装代码时，该代码的责任人可以更换他使用的公钥，而这实际上也就是更换他使用的密钥对。

2．代码的可信更新

根据约定 8.4，在更新第 L 层代码前，第 L 层代码是可以正常工作的，这意味着第 L 层代码已经顺利安装到 IBM 4758 中。根据推论 8.1，IBM 4758 已经拥有第 L 层代码的责任人的公钥，该公钥是可信，无须再进行认证。

可见，与代码安装相比，代码更新工作要简单一些，只需由代码的责任人认证要更新的代码的真实性。代码更新的方法如下。

方法 8.2　对于任意的 L（$1 \leqslant L \leqslant 3$），按照以下步骤更新第 L 层的代码。

（1）由第 L 层的责任人向 IBM 4758 签发更新第 L 层代码的命令：

$$\{C_{\text{update}}(L,\ \text{CODE}(L),\ K_{\text{PUB-new}}(L),\ \cdots)\}K_{\text{PRV}}(L) \qquad ④$$

其中，

$$C_{\text{update}}(L,\ \text{CODE}(L),\ K_{\text{PUB-new}}(L),\ \cdots) \qquad ⑤$$

表示“更新第 L 层代码”的命令，CODE(L) 表示第 L 层代码，$K_{\text{PUB-new}}(L)$ 表示第 L 层的责任人在代码更新后拟采用的新公钥，它可以与原来的公钥 $K_{\text{PUB}}(L)$相同。

（2）由 IBM 4758 的辅引导代码从闪存中获取第 L 层的责任人的公钥 $K_{\text{PUB}}(L)$，据此可以验证④式中的签名，从而获得如⑤式所示的“更新第 L 层代码”的命令。

（3）由辅引导代码根据⑤式所示的命令更新第 L 层代码。

图 8.10 中的带箭头的虚线描述代码更新时的认证思想：只需要由代码的责任人认证要更新的代码的真实性。

同样，由方法 8.2 的⑤式可见，在更新代码时，该代码的责任人也可以更换他使用的公钥，进而更换他使用的密钥对。

8.3.4　系统的可信引导

利用硬件实现的 DES 密码运算器，可以得到很好的 DES 密码运算支持，因此，IBM 4758 采用基于 DES 的消息认证码（MAC，Message Authentication Code）验证代码的完整性。

IBM 4758 在存储每一层代码 CODE(*L*)的同时，也存储该代码的消息认证码 MAC(*L*)。消息认证码 MAC(*L*)可用于验证代码 CODE(*L*)的完整性。

例 8.13　试说明利用代码 CODE(*L*)的消息认证码 MAC(*L*)验证代码 CODE(*L*)的完整性的方法。

解答：根据代码 CODE(*L*)计算出该代码的消息认证码 $MAC_{now}(L)$，设 IBM 4758 中存储的与代码 CODE(*L*)对应的消息认证码为 $MAC_{saved}(L)$，根据以下等式判断代码 CODE(*L*)的完整性

$$MAC_{now}(L) == MAC_{saved}(L)$$

如果以上等式成立，则认为代码 CODE(*L*)是完整的，否则，认为代码 CODE(*L*)是不完整的。

在安装或更新代码时，代码的责任人随代码一起提供了代码的消息认证码，即，方法 8.1 的③式和方法 8.2 的⑤式中都含有代码 CODE(*L*)的消息认证码 MAC(*L*)，用于安装或更新时的代码完整性检查，并供 IBM 4758 保存。

IBM 4758 在进行可信引导的过程中，使用基于 DES 的消息认证码检查代码的完整性，其引导过程如下：

（1）IBM 4758 硬件在上电或复位时，自动执行存储在 ROM 中的主引导程序 miniboot0，此时，硬件锁的状态值被置为 0；

（2）首先，miniboot0 执行初始上电自检程序 POST0，检查基本硬件的工作状况，然后，miniboot0 验证它自己的消息认证码和辅引导程序 miniboot1 的消息认证码，如果发现问题，则中止引导过程，如果顺利，则促使硬件锁的状态值加 1，启动 miniboot1 的执行；

（3）首先，miniboot1 执行后续上电自检程序 POST2，检查其他硬件的工作状况，然后，miniboot1 验证第 2 层和第 3 层的消息认证码，如果发现问题，则中止引导过程，如果顺利，则促使硬件锁的状态值加 1，启动第 2 层（即操作系统）的执行；

（4）操作系统进行初始化处理，进入正常工作状态后，如果必要，则促使硬件锁的状态值加 1，然后，启动第 3 层（即应用软件）的执行；

（5）应用软件开始运行，必要时，它也可以促使硬件锁的状态值加 1，以保护它的私有信息。

可以把 LBBRAM 中的第 0、1、2、3 号区域分别分配给第 0、1、2、3 层代码使用，把闪存中受保护的第 1、2、3 号区域分别分配给第 1、2、3 层代码使用，通过以上引导过程中对硬件锁的控制，可以使各层代码及其信息得到有效的保护。

8.3.5　系统完整性保障措施

本节介绍的 IBM 4758 安全协处理器是一个物理上密封的缩微计算机，它必须插接到主机系统的 PCI 总线上才能进行工作，但是，它具有一定的独立性，能够运行独立的应用软件。本节的主要目的是了解通过把系统部署到密封的安全协处理器中而使系统的完整性得到保护的方法。

8.3.4 节介绍了 IBM 4758 采用消息认证码检查系统完整性的方法，但是，必须注意：基于消息认证码的完整性检查并不是 IBM 4758 中的唯一的完整性保护措施。

IBM 4785 提供了物理攻击反应、硬件锁访问控制、代码装入认证和消息认证码检查等多种功能，通过多种措施的综合运用来实现系统的完整性保护。

物理攻击反应功能可以感应出物理攻击的企图，并做出以防止秘密信息泄露为主的反应。硬件锁访问控制功能通过硬件锁的状态控制不同存储空间区域的访问许可，防止系统代码和信息被非法篡改。代码装入认证功能在代码安装和更新阶段进行严格把关，确保装入到协处理器中的代码的真实性。消息认证码检查功能主要针对硬件存储介质中存在的出错概率，防止随机的硬件差错影响系统的完整性。

8.4　基于安全中央处理器的完整性

在计算机系统中，如果只有中央处理器是值得信赖的，除此之外，所有的硬件和软件（包括内存和操作系统在内）都是不可信的，那么，如何为系统提供完整性支持呢？

本节以美国麻省理工学院（MIT，Massachusetts Institute of Technology）提出的 AEGIS 模型为例，介绍解决这个问题的方法。请注意：8.2 节介绍的安全引导模型也称为 AEGIS，那是由美国宾夕法尼亚大学（University of Pennsylvania）提出的，与这里所讨论的模型没有任何联系，它们取名相同，只是巧合。本节的讨论特指 MIT 的 AEGIS 模型。

8.4.1　进程完整性验证框架

MIT 的 AEGIS 模型是一个基于安全中央处理器的完整性验证模型，它以进程的完整性为主要对象，以普通中央处理器为基础，在其中增加安全处理单元，从而提供对进程的完整性进行验证的功能。模型的目标是发现或防止所有可能影响进程行为的篡改事件，不管是通过物理手段的篡改还是通过软件手段的篡改。

AEGIS 模型认为，进程的完整性取决于进程在初始状态中的完整性、在中断过程中的完整性、呈现于存储介质时的完整性及输出结果的完整性等。因此，该模型主要在这

些方面提供相应的完整性支持。模型采用的完整性验证方法是基于哈希值的方法。

在本节的讨论范围内，AEGIS 模型定义了两种进程模式：普通模式和篡改响应模式。普通模式下的进程不需要进行完整性验证，篡改响应模式下的进程才需要进行完整性验证。

当进程从普通模式转入篡改响应模式时，进程开始进入完整性验证意义下的初始状态。进程的初始状态启动时，系统的可信计算基（TCB，Trusted Computing Base）计算并检查进程对应的程序的哈希值，同时，检查进程运行环境的纯净性，以确保进程的运行从良好的状态开始。也就是说，初始的进程是完整的，它的运行环境没有被非安全因素污染过。

进程在运行的过程中经常会遇到中断事件，中断发生时，系统要进行环境切换。进程的环境信息属于进程的一部分，进程的完整性包括进程的环境信息的完整性。在中断处理的环境切换过程中，AEGIS 模型通过保护进程的寄存器信息来保护进程环境信息的完整性。

进程的代码和数据是驻留在存储介质中的，这里涉及的存储介质的类型包括中央处理器中的 Cache 高速缓存、中央处理器外的内存和磁盘等。系统需要用到进程的代码或数据时，首先把它们装入到内存中，进而装入到 Cache 高速缓存中，然后执行相应的代码并处理相应的数据。在虚拟内存管理中，内存中的页面有可能被唤出到交换区中，交换区位于磁盘上，此时，在被唤出的页面上的代码或数据驻留被放置到磁盘上。

Cache 高速缓存属于片上存储介质，内存和磁盘属于片外存储介质。位于片上和片外存储介质上的代码和数据的完整性都需要得到保护。AEGIS 模型假设中央处理器能够抗击物理攻击，因而，片上存储介质只可能遭受软件攻击，但片外存储介质有可能遭受物理攻击和软件攻击。当系统从片外存储介质上把一个存储块读入片内存储介质时，AEGIS 模型的 TCB 验证该存储块的完整性。

AEGIS 模型设有专门的完整性验证机制，用于验证存储块的完整性。对于任何一个需要进行完整性保护的内存位置，完整性验证机制只允许一个进程对它进行合法的修改操作。显然，如果整个内存空间都需要进行完整性保护，篡改响应模式下的不同的进程将无法共享任何内存区域。

为了解决共享内存的问题，AEGIS 模型把进程的虚拟内存空间划分为两大部分，一部分需要进行完整性验证，另一部分不需要进行完整性验证。其中，最高地址位为 1 的地址定义为需要进行完整性保护的地址；最高地址位为 0 的地址定义为不需要进行完整性保护的地址。

例 8.14　设 AEGIS 模型中的进程 A 在篡改响应模式下运行，它需要读取虚拟地址 ADD_1 和 ADD_2 中的数据，已知 ADD_1 的最高位是 1，ADD_2 的最高位是 0，试问读 ADD_1 和 ADD_2 中的数据时，TCB 进行的处理有什么不同？

解答：由于虚拟地址 ADD_1 处的数据需要进行完整性验证，所以，读 ADD_1 对应的数据块时，TCB 中的完整性验证机制对数据块进行完整性验证。由于虚拟地址 ADD_2 处的数据不需要进行完整性验证，所以，读 ADD_2 对应的数据块时，TCB 中的完整性验证机制不对数据块进行完整性验证。

初始状态中的完整性保护、中断处理时的完整性保护和驻留在存储介质中的进程代码和数据的完整性，可以保证进程能够正确地运行，但是，还不能保证接收者接收到的来自该进程的结果信息是可信的。

为了使接收者能够获得进程产生的可信的结果信息，AEGIS 模型的 TCB 对进程产生的结果信息进行数字签名，然后再传送给接收者。

例 8.15　设在篡改响应模式下运行的进程 A 对应的程序为 Prog，该进程产生的结果信息为 M，中央处理器的私钥/公钥对中的私钥为 $K_{\text{PRV-CPU}}$，试给出 TCB 对结果信息 M 进行数字签名的方法。

解答：设 TCB 在进程 A 的初始状态计算得到的程序的 Prog 的哈希值为 $H(\text{Prog})$，则 TCB 对进程 A 产生的结果信息进行数字签名得到的结果为

$$\{H(\text{Prog}), M\}K_{\text{PRV-CPU}}$$

例 8.15 的数字签名结果中的 $K_{\text{PRV-CPU}}$ 证明了特定中央处理器的身份，$H(\text{Prog})$证明了特定程序的身份，而 M 证明了信息的真实性。一个特定中央处理器的身份可以隐含地证明一个特定系统的身份，所以，当一个接收者收到含有该数字签名的信息时，它能断定该信息是由特定的系统（系统认证）运行特定的程序（程序认证）得到的特定结果信息（信息认证）。

AEGIS 模型的完整性验证框架通过完整性验证和保护及数字签名措施，确保系统中运行的进程能够保持良好的完整性，并把产生的结果信息可信地传送给相应的接收者。模型通过在中央处理器中添加安全支持单元来实施这些措施。添加安全支持单元后设计出的安全中央处理器及相应的系统的结构原理如图 8.11 所示。

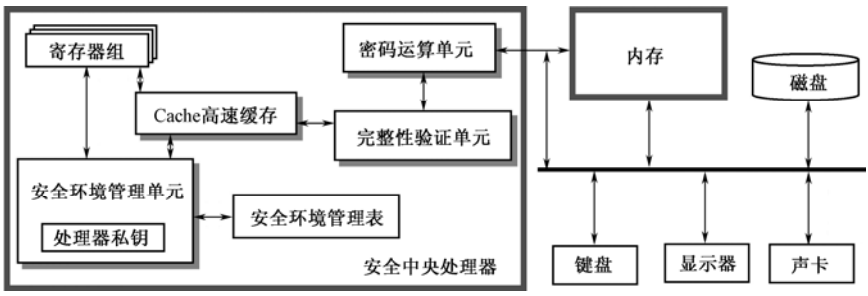


图 8.11　安全中央处理器及系统结构原理

添加的安全支持单元主要包括密码运算单元、完整性验证单元、安全环境管理单元

及安全环境管理表等，8.4.2 节首先介绍完整性验证单元。

8.4.2　完整性验证单元

安全中央处理器中的完整性验证单元的主要功能是验证进程在片外存储介质中的信息（包括代码和数据）的完整性，验证工作是针对虚拟地址空间实施的，所以，这可以抽象为验证进程的虚拟内存空间的完整性。

完整性验证单元实现了对 8.1.3 节中介绍的莫科尔树模型的支持，按照该模型的方法实施完整性验证。实施莫科尔树模型要解决的两个重要问题是建立莫科尔树和利用该树进行完整性验证。完整性验证单元利用片外存储介质存放莫科尔树，借助片上 Cache 高速缓存实施完整性验证操作。

一个进程的整个虚拟内存空间可以按照一定方式划分成一系列存储块，以这些存储块为树叶，可以构造出一棵莫科尔树。利用这棵莫科尔树，完整性验证单元可以方便地对进程的虚拟内存空间中的存储块进行完整性验证。

当处理器欲从进程的虚拟内存空间中把信息读到 Cache 高速缓存中时，完整性验证单元验证指定虚拟地址对应的存储块的完整性，验证通过后，处理器才把相应的存储块中的信息存入到 Cache 高速缓存中。

例 8.16　设在篡改响应模式下运行的进程 A 要读取其虚拟地址 V_{ADD1} 处的数据，试说明安全中央处理器读取该数据的过程。

解答：设虚拟地址 V_{ADD1} 对应的物理地址为 P_{ADD1}，物理地址 P_{ADD1} 对应的存储块为 M_{block1}。如果 V_{ADD1} 的最高位是 1，则完整性验证单元验证存储块 M_{block1} 的完整性，如果完整性良好，则处理器把该存储块读到 Cache 高速缓存中。如果 V_{ADD1} 的最高位是 0，则完整性验证单元不必进行完整性验证，处理器直接把存储块 M_{block1} 读到 Cache 高速缓存中。

系统为每个进程定义一棵莫科尔树，为每棵莫科尔树分配一个独立的虚拟内存空间。这样，一个进程对应两个虚拟内存空间：一个是进程的程序（包含代码和数据）的虚拟内存空间，另一个是进程的莫科尔树的虚拟内存空间。完整性验证单元以莫科尔树虚拟内存空间为支撑，实现对程序虚拟内存空间存储块的完整性验证。

为了减少完整性验证所产生的性能开销，完整性验证单元在验证过程使用 Cache 高速缓存。系统将部分 Cache 高速缓存用于暂存进程莫科尔树的树节点，每个树节点占用一个 Cache 块。Cache 块暂存树节点的哈希值，同时记录树节点的有效位和虚拟地址。

树节点的有效位用于标识树节点的有效性，1 表示已经有效，0 表示尚未有效。对于新建立的莫科尔树，所有树节点的有效位均设为 0。等到完整性验证过程中需要用到某个树节点时，再计算它的哈希值，并把它的有效位设为 1。

完整性验证单元对程序的存储块进行完整性验证的一般思路是按照从子节点到父节点的方向，沿着从树叶到树根的路径，由下至上逐层验证，一直到树根节点为止。完整

性验证方法如下。

方法 8.3　完整性验证单元验证任意一个给定的树节点（包括树叶节点）的方法是首先把待验证的节点作为当前节点，然后执行以下步骤：

（1）从莫科尔树的虚拟内存空间中找到当前节点的兄弟节点；

（2）把当前节点和它的兄弟节点的信息连接起来；

（3）计算连接结果的哈希值；

（4）找到当前节点的父节点的哈希值；

（5）检查第（3）步计算得到的哈希值与第（4）步找到的哈希值是否相同，如果不同，则报告完整性失效，结束；

（6）如果当前节点的父节点是树根，则结束，否则把它作为当前节点，转到第（1）步。

为了便于找到给定树节点的父节点，在虚拟内存空间中可以按照宽度优先的方式组织莫科尔树的结构。Cache 块中记录的树节点的虚拟地址的作用之一就是便于找到该节点的父节点的虚拟地址。

采取了用 Cache 块暂存树节点的策略后，完整性验证无须一直延伸到树根，只须持续到存在于 Cache 块中的第一个树节点即可。相应的验证方法如下。

方法 8.4　在以 Cache 块暂存树节点的前提下，完整性验证单元验证任意一个给定的树节点（包括树叶节点）的方法是首先把待验证的节点作为当前节点，然后执行以下步骤：

第（1）步至第（5）步与方法 8.3 的相同；

（6）如果当前节点的父节点在 Cache 块中或是树根，则结束，否则把它作为当前节点，转到第（1）步。

实现基于莫科尔树模型的机制需要考虑树节点的更新问题，完整性验证单元采用如下更新方法。

方法 8.5　完整性验证单元对任意给定的树节点进行更新的方法是首先把待更新的节点作为当前节点，检查它的完整性，在完整性良好的情况下，执行以下步骤：

（1）修改当前节点；

（2）根据当前节点及其兄弟节点的信息重新计算并更新当前节点的父节点；

（3）如果当前节点的父节点是树根，则结束，否则把当前节点的父节点作为当前节点，转到第（2）步。

对于某个给定进程而言，完整性验证单元的功能从进程的初始状态起开始发挥作用。即，当进程从普通模式进入篡改响应模式时，处理器计算进程的程序的哈希值，为

进程分配用于莫科尔树的虚拟内存空间，并在该虚拟内存空间中建立相应的莫科尔树。此时，还无须确定各节点的哈希值，各节点处于尚未有效的状态，有效位的值为 0。

在完整性验证过程中，当完整性验证单元读取 Cache 块中的树节点时，如果发现该节点的有效位是 0，则首先找到它的子节点对应的 Cache 块，再根据子节点对应的 Cache 块计算出它的哈希值，然后把它的有效位设为 1。

完整性验证单元按照方法 8.4 对进程的程序的虚拟内存空间的存储块的完整性进行验证。Cache 块记录着对应节点的虚拟地址，在验证过程中，结合宽度优先的树结构，根据当前节点的虚拟地址可以计算出该节点的父节点的虚拟地址，虚拟地址可以翻译成物理地址，有了父节点的物理地址便可得到父节点的哈希值，因而，根据给定节点可以确定其父节点的信息，从而，验证过程可以不断向前推进，直到完成验证任务。

8.4.3　硬件支持的完整性验证

在前面两个小节的基础上，本小节介绍 AEGIS 模型通过安全中央处理器中的专用硬件单元，从初始状态完整性支持、中断过程完整性支持、片上 Cache 的完整性支持、片外存储介质的完整性支持、结果信息的完整性支持等多个侧面，综合提供完整性支持的方法。

为了支持篡改响应模式下的完整性保护，AEGIS 模型的安全中央处理器提供以下 4 条专用指令供应用程序使用。

（1）enter_aegis：使进程从普通模式转入篡改响应模式；

（2）exit_aegis：使进程从篡改响应模式转入普通模式；

（3）sign_msg：使用处理器的私钥对程序的哈希值和给定的信息进行数字签名；

（4）get_random：由安全硬件随机数生成器生成一个随机数。

安全中央处理器的安全环境管理单元用于为进程提供安全的运行环境。安全环境管理单元为每个在篡改响应模式下运行的进程分配一个非 0 的安全进程身份标识（SPID）。普通模式下运行的进程的 SPID 取值为 0。处理器中设立了安全环境管理表，由安全管理单元用于记录进程的环境相关信息。

对于在篡改响应模式下运行的每一个进程，安全环境管理单元在安全环境管理表中为它分配一个记录，记录的内容包括进程的 SPID、程序的哈希值、寄存器组中各寄存器的值、莫科尔树的树根的值等。

当一个进程执行 enter_aegis 指令时，安全环境管理单元在安全环境管理表中为它创建相应的记录。当一个进程执行 exit_aegis 指令时，安全环境管理单元删除它在安全环境管理表中的相应记录。操作系统在杀死一个进程时也可以删除该进程在安全环境管理表中的记录。

安全环境管理表存放在片外的虚拟内存空间中，它的完整性由完整性验证单元保护。处理器用一个专用的片上 Cache 来存放当前进程在安全环境管理表中的记录。

进程通过执行 enter_aegis 指令，由普通模式转入篡改响应模式，该指令执行时，安全环境管理单元计算进程的程序的哈希值，并检查进程运行环境的纯净性。从安全环境管理单元计算程序的哈希值的那一刻开始，程序的完整性即受到片上和片外完整性保护机制的保护。

在中断响应方面，当进程在运行过程中遇到中断时，安全环境管理单元在安全环境管理表中保存该进程的运行环境中的所有寄存器信息。中断结束时，安全环境管理单元根据安全环境管理表中保存的寄存器信息恢复该进程的运行环境。这样，可以在中断处理过程中保护进程运行环境的完整性。

进程在片上 Cache 中的完整性借助 Cache 标记提供支持。使用 Cache 时，要注明哪个进程借助该 Cache 块访问哪个虚拟地址上的信息，方法是在 Cache 块中记录进程的身份（由 SPID 表示）和所存信息的虚拟地址，即用于暂存进程的程序信息的每个 Cache 块中包含的内容有程序信息、进程的 SPID 和程序信息的虚拟地址等。

例 8.17　设进程 A 的安全进程标识为 $SPID_1$，该进程的程序虚拟内存空间的虚拟地址 V_{ADD1} 对应的存储块为 M_{block1}，该存储块暂存在 Cache 块 C_{block1} 中，则 C_{block1} 中包含的信息有 M_{block1}、$SPID_1$ 和 V_{ADD1}。

例 8.17 说明身份为 $SPID_1$ 的进程拥有 Cache 块 C_{block1}，该 Cache 块中暂存的是虚拟地址 V_{ADD1} 所对应的存储块。

当在篡改响应模式下运行的一个进程访问一个需要完整性保护的地址时，处理器在使用一个相应的 Cache 块之前，首先对它进行合法性检查，如果进程的 SPID 与该 Cache 块中记录的 SPID 相同，而且，待访问的虚拟地址与该 Cache 块中记录的虚拟地址相同，则合法性检查得以通过，进程可以直接访问该 Cache 块。否则，系统根据待访问的地址更新该 Cache 块，完整性验证单元验证该 Cache 块的完整性，并更新该 Cache 块中记录的 SPID 和虚拟地址。此后，进程才能够访问相应的 Cache 块。

进程在片外存储介质中的完整性由完整性验证单元提供支持。由于完整性验证单元对虚拟内存空间进行保护，所以，不管进程信息是存放在片外内存中，还是由于页交换的原因被存放在用于内存交换区的磁盘上，该信息的完整性都可以得到保护。

进程通过执行 sign_msg 指令对自己产生的输出结果进行数字签名，从而在进程产生的结果信息传送给接收者的过程中，保护结果信息的真实性和完整性。

8.5　内核主导的完整性度量

系统完整性研究的重要思想之一是由硬件提供完整性的根，并建立从完整性的根到应用软件的完整性链，从而保护应用软件系统的完整性。前面的 8.2 节和 8.3 节都体现了这样的思想。不过，那里的讨论把操作系统和应用软件作为单一的整体看待，而实际

上，两者都有复杂的结构，它们的完整性往往很难以单一整体的方式进行度量。

本节以 IBM 沃森研究中心（IBM T.J. Watson Research Center）提出的完整性度量体系结构（IMA，Integrity Measurement Architecture）为例，以操作系统内核和用户空间的进程为对象，讨论对它们进行完整性度量的基本方法。

8.5.1　完整性度量对象的构成

IMA 是一个基于 Linux 操作系统的模型。在 Linux 操作系统框架下，系统空间划分为内核空间和用户空间两个部分，操作系统内核在内核空间运行，其他程序以进程的形式在用户空间运行。

操作系统内核和用户空间的进程是 IMA 模型完整性度量的主要对象，其中，用户空间的进程包括操作系统的服务进程和应用软件的进程。普通 Linux 操作系统中的内核和进程都不是单一的实体，无法把它们作为单一整体进行完整性度量。

内核可以划分为基本内核和可装载内核模块。进程的程序可以划分为基本程序和可扩展程序，可扩展程序可以以动态库和动态模块等形式出现。基本内核和基本程序都可以作为单一整体进行完整性度量，它们的完整性度量分别是内核和进程完整性度量的基础。

程序的内容可以分为代码和数据，数据又可以分为结构化的静态数据和非结构化的动态数据。要度量程序的完整性，必须同时度量代码的完整性和代码所处理的数据的完整性。

例 8.18　在 Linux 操作系统环境下，基于 Apache Web 服务器和 Tomcat Web 容器开发的一个在线售书应用系统的服务器端的系统构成如图 8.12 所示。内核完整性度量的对象包括基本内核和可装载内核模块，进程完整性度量的对象包括基本可执行程序、动态库、动态模块、静态数据和动态数据等。

其中，影响系统完整性的用户空间的进程的可执行程序代码包括：

- Apache 服务器（apachectl, httpd, …）；
- Apache 模块（mod_access.so, mod_auth.so, mod_cgi.so, …）；
- Tomcat servlet 机器（startup.sh, catalina.sh, java, …）；
- 动态库（libjvm.so, libcore.so, libjava.so, libc2.3.2.so, libssl.so.4, …）。

由应用程序装载的影响系统完整性的文件包括：

- Apache 配置文件（httpd.conf）；
- Java 虚拟机安全配置（java.security, java.policy）；
- Servlet 和 Web 服务库（axis.jar, servlet.jar, wsdl4j.jar, …）。

应用程序在运行过程中涉及的影响系统完整性的关键动态数据包括：

- 来自远程客户、管理员和其他 Servlet 的各种请求；
- 图书订单数据库。

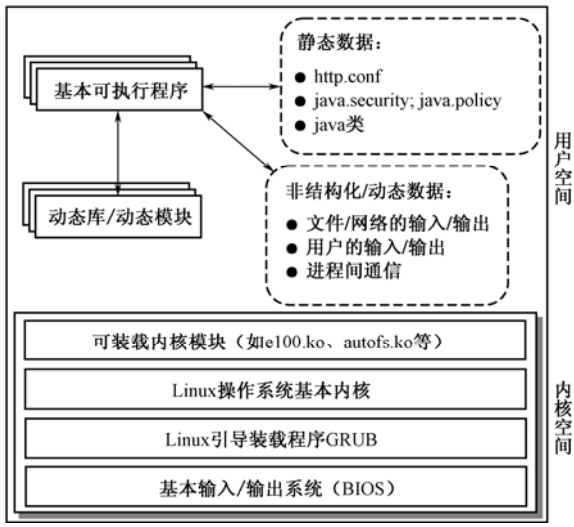


图 8.12　一个在线售书应用系统的服务器端系统组成

对于以上两类动态数据，需要确定以下事情：

- 是不是只有高完整性的进程才能修改订单数据或管理员命令（毕巴模型）；
- 低完整性的请求是可以被转换成高完整性的请求还是被拒绝（克-威模型）。

操作系统的基本内核由操作系统的引导装载程序装入内存并启动运行，装入前，引导装载程序可以度量它的完整性。

在 Linux 操作系统中，启动一个基本可执行程序的运行的方法是：

- 根据该可执行程序文件的格式装载一个合适的程序解释器（即动态装载器，如 ld.so 等）；
- 由已装入内存运行的动态装载器装载该可执行程序的代码和相应的支持库。

动态装载器在装载基本可执行程序的代码的过程中，运用可执行标记来把相关的文件映射成内存中的可执行程序，所以，当基本可执行程序代码被装载时，内核是知道的。

可装载内核模块的情况与此有所不同，它们是由诸如 modprobe 或 insmod 的应用程序装载的，它们是在已经被装载到内存中之后才映射成内存中的可执行程序的，所以，当应用程序把它们从文件系统中装载到内存中时，内核并不知道。

可执行脚本是应用程序中的另一种常见的典型构成，内核很难知道它们什么时候被装载，它们是以普通文件的形式被装载到脚本解释器（如 bash 等）中的，由脚本解释器解释执行。

应用程序在运行过程中还可能装载一些其他类型的文件，例 8.12 给出了这些类型的文件的一些例子，这样的文件什么时候被装载，内核也是很难知道的。

完整性度量对象在装载方面不容易确定的特点给完整性度量带来了一定的困难。非结构化动态数据的特点则进一步增加了完整性度量的难度。结合上面的讨论，一个系统中需要进行完整性度量的对象可以归纳为可执行内容、结构化数据和非结构化数据等类

型，IMA 模型主要考虑在操作系统中为程序代码和结构化数据的完整性度量提供支持的基本方法。

8.5.2　完整性度量的基本机制

IMA 模型以可信平台模块（TPM，Trusted Platform Module）硬件芯片为基础来构造系统完整性的度量方法。IMA 模型将 TPM 硬件芯片作为完整性度量的根，按照以下的基本思路确定操作系统基本内核的完整性：

TPM → BIOS → 引导装载程序 → 基本内核

作为完整性度量的根，在默认情况下，TPM 的完整性是良好的。TPM 度量 BIOS 的完整性，BIOS 度量引导装载程序的完整性，引导装载程序度量基本内核的完整性。

IMA 模型完整性度量的基本机制从两个方面进行完整性度量：一方面，度量基本可执行程序的完整性；另一方面，度量其他可执行内容的完整性和敏感数据文件的完整性。度量的基本思想是：

（1）首先度量操作系统基本内核的完整性；

（2）基本内核度量演变后的内核的完整性（演变源自可装载内核模块的装载）；

（3）内核创建用户空间的进程；

（4）内核度量装载到进程中的可执行代码的完整性，如内核度量动态装载器和 httpd 的完整性；

（5）以上可执行代码度量后续装载的安全敏感输入的完整性，如 httpd 度量配置文件或可执行脚本的完整性。

IMA 模型利用 TPM 芯片进行完整性度量的基本方法是通过 SHA1 运算对需度量的某个文件的整体内容计算哈希值，得到的结果是一个 160 位的哈希值，用于作为待度量文件的指纹。

例 8.19　设 file1 是一个待度量的文件，则 IMA 模型度量它的完整性的方法是进行以下计算：

$$H_{\text{file1}} = \text{SHA1(file1)}$$

其中，H_{file1} 是得到的 160 位的值，是文件 file1 的指纹。

在整个系统的完整性度量过程中，IMA 模型通过 TPM 芯片的 TPM_extend 功能把每次度量得到的文件指纹合成到 TPM 芯片中的某个平台配置寄存器（PCR，Platform Configuration Register）中，合成的方法是把 PCR 寄存器原来的值与文件指纹值连接起来，再进行 SHA1 运算，得到的结果作为该 PCR 寄存器新的值。

例 8.20　设 IMA 模型用 TPM 芯片中的第 i 个 PCR 寄存器记录系统完整性度量过程的结果信息，H_{file1} 是文件 file1 的指纹，则把该文件的度量结果合成到 PCR 寄存器中的方法是：

$$\text{TPM_extend (PCR}[i], H_{\text{file1}})$$

其产生的效果是：

$$\text{PCR}[i] = \text{SHA1}(\text{PCR}[i] \parallel H_{\text{file1}})$$

由每次度量得到的文件指纹的合成而生成的 PCR 寄存器的值是完整性度量全过程的指纹，唯一地标识了到某个时刻为止完整性度量过程的最终结果。不同的度量过程对应不同的指纹。

除了利用 PCR 寄存器标识完整性度量过程的最终结果以外，IMA 模型在操作系统中设立了一张完整性度量表，用于记录每次度量时产生的文件指纹，文件内容的完整性度量如图 8.13 所示。

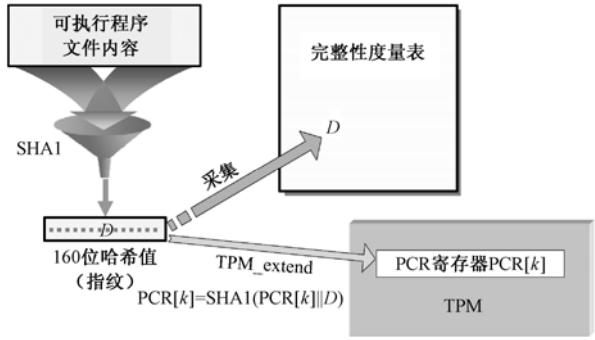


图 8.13　文件内容的完整性度量

利用完整性度量表中的文件指纹，可以很容易地计算出度量过程的最终指纹。PCR 寄存器的安全性受到了硬件的保护，其中的值是可信的。通过对比计算出的最终指纹和相应 PCR 寄存器中的值，可以验证完整性度量表的完整性，如果两值相等，则表明完整性度量表是完整的，否则表明完整性度量表已遭篡改。

例 8.21　设在 T_i 时刻，IMA 模型已经依次对 file1、file2、…、file*i* 等 *i* 个文件的完整性进行了度量，那么，此刻，完整性度量表 M_{list} 的内容为

$$M_{\text{list}} = \{H_{\text{file1}}, H_{\text{file2}}, \cdots, H_{\text{file}i}\}$$

用于记录完整性度量过程结果信息的 PCR 寄存器 PCR[*k*]的值表示的意义是：

$$\text{PCR}[k] = \text{SHA1}(\cdots\ \text{SHA1}(\text{SHA1}(0 \parallel H_{\text{file1}}) \parallel H_{\text{file2}})\ \cdots\ \parallel H_{\text{file}i})$$

根据完整性度量表 M_{list} 容易计算出：

$$H_{T_i} = \text{SHA1}(\cdots\ \text{SHA1}(\text{SHA1}(0 \parallel H_{\text{file1}}) \parallel H_{\text{file2}})\ \cdots\ \parallel H_{\text{file}i})$$

如果 H_{T_i} 与 PCR[*k*]相等，那么，在 T_i 时刻，完整性度量表 M_{list} 的完整性是良好的，否则完整性度量表 M_{list} 的完整性已经受损。

操作系统中的完整性度量表 M_{list} 在 T_i 时刻具有良好的完整性，说明该表能够反映从上电时刻到 T_i 时刻系统完整性度量的真实情况，也就是说，表中的度量结果是可信的，能够体现系统的真实状态。

那么，依据完整性度量表 M_{list} 是否已经可以判断系统是否可信了呢？显然还不行。IMA 模型还需要另外一张完整性度量表 M_{trusted}，该表是在已知系统可信的情况下生成的，并且，其完整性是良好的，因而，它能够反映可信系统的真实状态。

表 M_{list} 是实际系统的完整性度量表，表 M_{trusted} 是可信系统的完整性度量表。通过检查表 M_{list} 与表 M_{trusted} 是否一致，可以判断实际系统在 T_i 时刻的完整性是否良好，即实际系统是否可信，IMA 模型完整性度量框架如图 8.14 所示。IMA 模型依靠系统以外的其他手段来保护表 M_{trusted} 的完整性。

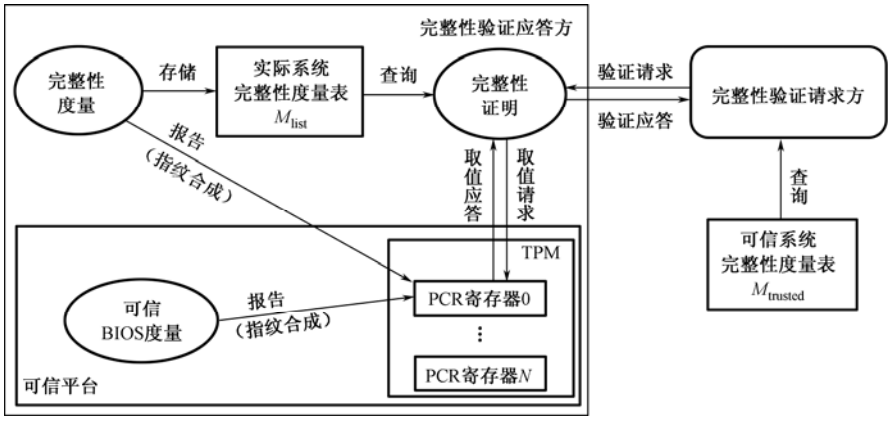


图 8.14　IMA 模型完整性度量框架

完整性度量机制在系统运行过程中对系统进行完整性度量，产生的结果包括实际系统的完整性度量表 M_{list} 和相应的 PCR 寄存器的值。当系统接收到完整性验证请求时，完整性证明机制向验证的请求方提供系统完整性的证明。根据应答方提供的证明信息和可信系统的完整性度量表 M_{trusted}，完整性验证的请求方验证实际系统的完整性。

8.5.3　完整性度量的实现方法

从计算机上电开始，到操作系统基本内核开始运行之前，系统处于引导阶段。学习了本章前面的内容之后，一定对系统引导阶段的完整性度量有了比较清楚的认识。引导阶段结束之时，基本内核的完整性已经得到了保障。在介绍下面的内容之前，假设已经知道基本内核的完整性是良好的。

8.5.1 节和 8.5.2 节介绍了 IMA 模型完整性度量的整体框架，本小节将深入讨论操作系统内核完整性度量和用户空间进程完整性度量的实现方法。

由于基本内核已经具有良好的完整性，所以，这里主要讨论可装载内核模块完整性度量的实现方法。用户空间进程完整性度量的实现方法主要从基本可执行程序、动态可装载库和可执行脚本等方面进行讨论。

Linux 操作系统可装载内核模块是由用户空间的 modprobe 或 insmod 程序装载的，装载和完整性度量如图 8.15 所示。

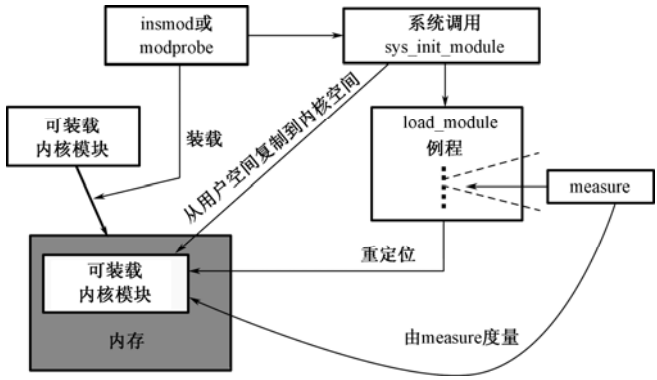


图 8.15　可装载内核模块的装载和完整性度量

用户空间的 modprobe 或 insmod 程序把可装载内核模块装载到内存的用户空间中，进而调用系统调用 sys_init_module，通知内核：“有新的可装载内核模块加入内核”。

系统调用 sys_init_module 的重要任务之一是把已装到内存中的可装载内核模块从用户空间复制到内核空间，它调用内核例程 load_module 对可装载内核模块进行重定位。

可以把内核例程 load_module 中的重定位代码之前的位置定义为度量点，在该位置上对可装载内核模块进行完整性度量。可行的方法是设计一个 measure 函数，用于执行完整性度量任务，并在内核例程 load_module 的度量点处插入一个调用 measure 函数的语句。

当内核例程 load_module 执行到调用 measure 函数的语句时，measure 函数被执行，它对可装载内核模块所在的内存区域进行完整性度量。这样，内核在对可装载内核模块进行重定位之前，便可以完成对该可装载内核模块的完整性度量。

不管是基本可执行程序还是动态可装载库，可执行程序文件装载到内存中时，要经过内存映射，映射成可执行代码后才能运行。在 Linux 操作系统中，针对这样的映射，有一个 LSM 钩子 file_mmap 和它对应。IMA 模型利用 file_mmap 钩子实现对可执行程序的完整性度量。

在 Linux 操作系统中，用户空间进程通过调用系统调用 execve，启动基本可执行程序的装载，装载和完整性度量如图 8.16 所示。

系统调用 execve 执行时，内核调用二进制代码处理例程，根据基本可执行程序二进制代码的类型定位合适的用户空间的装载器。内核把相应的装载器映射到内存中，并设置相应的运行环境，使系统调用 execve 返回后，该装载器即开始运行。

该装载器把用户空间的相应可执行程序装载到内存中，其间，内核进行内存映射，把该程序映射成可执行代码，最后，装载器把控制权传给可执行程序，可执行程序开始运行。

内核进行内存映射前，会执行 file_mmap 钩子的钩子函数，因此，可以在该钩子函数中调用 measure 函数，对可执行程序进行完整性度量。这样，用户空间的基本可执行程序被装载时，在可执行代码的内存映射实施之前，完整性得到度量。

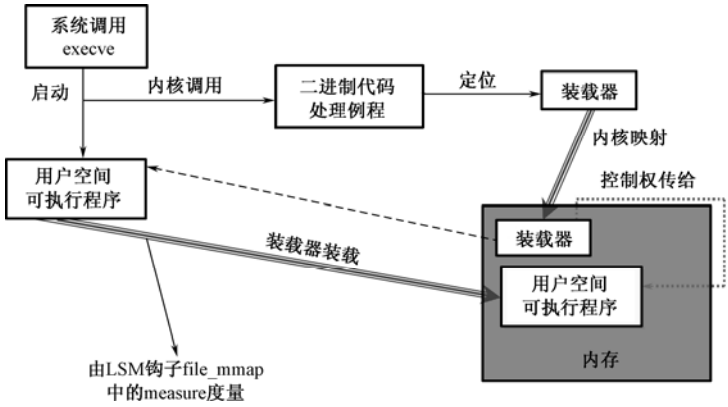


图 8.16　可执行程序的装载和完整性度量

用户空间的动态可装载库由用户空间的装载器进行装载，装载和完整性度量如图 8.17 所示，该装载操作对内核是透明的。

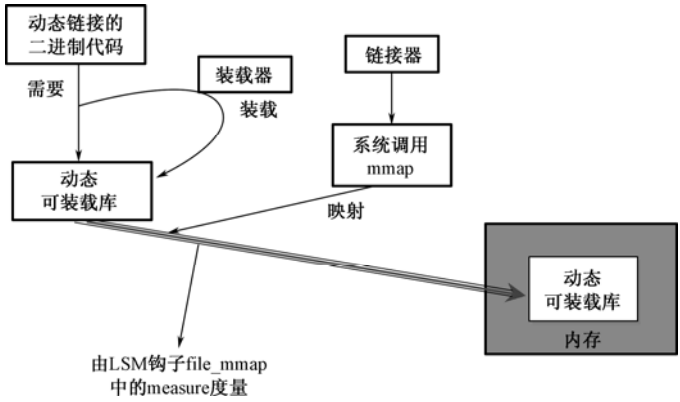


图 8.17　动态可装载库的装载和完整性度量

动态可装载库由装载器动态装载到内存后，要由链接器动态链接到相应的可执行代码中。链接器在工作过程中，调用系统调用 mmap 实施动态可装载库的内存映射。系统调用 mmap 会执行 file_mmap 钩子的钩子函数，因此，已经插入到该钩子函数中的 measure 函数调用也可以对动态可装载库进行完整性度量。

由此可见，插入到 file_mmap 钩子的钩子函数中的 measure 函数调用，既可以对基本可执行程序进行完整性度量，也可以对动态可装载库进行完整性度量。

可执行脚本由脚本解释程序装载和解释执行，它们是以普通文件的形式被装载的，所以，内核无法识别出它们的装载情况，装载和完整性度量如图 8.18 所示。

可以由脚本解释程序在装载可执行脚本前启动脚本的完整性度量操作。可行的方法是使用 sysfs 伪文件系统，即由用户空间的脚本解释程序把可执行脚本文件的文件描述符等信息写入/sys/security/measure 中，由 sysfs 伪文件系统调用 measure 函数对脚本文件

进行完整性度量。sysfs 伪文件系统接收的是文件描述符，measure 函数使用的是文件指针，所以，sysfs 伪文件系统根据文件描述符，通过 fget 例程找到相应的文件指针，然后调用 measure 函数。

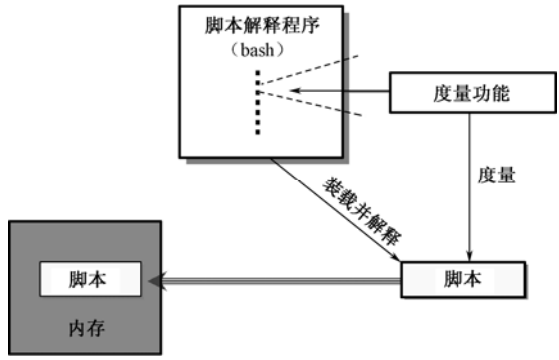


图 8.18　可执行脚本的装载和完整性度量

综上所述，以 TPM 硬件芯片和 Linux 操作系统为背景的 IMA 模型实现完整性度量的方法可以归纳为两个方面：其一是设计一个 measure 内核函数，用于对指定的内容进行完整性度量；其二是确定度量点，用于启动 measure 内核函数的完整性度量工作。measure 内核函数可以度量文件或内存区域的完整性。度量点设置在内核 load_module 例程、file_mmap 钩子函数和 sysfs 伪文件系统 /sys/security/measure 中。通过在内核空间和用户空间的度量点处启动内核主导的完整性度量操作，实现整个系统范围的操作系统内核和用户空间进程的完整性度量。

8.6　文件系统的完整性检查

前面从多个侧面介绍了借助硬件安全机制和操作系统内核进行系统完整性度量的典型方法。遵循从低层硬件到高层应用软件进行全方位考察的思路，下面讨论在应用软件的层面上进行系统完整性度量的代表性方法。在这方面，Tripwire 的完整性检查具有广泛的影响，本节以它为例进行讨论。

Tripwire 是对 UNIX 环境下的文件系统进行完整性检查的著名经典工具。1992 年，美国普渡大学的 G.H. Kim 和 E.H. Spafford 开发了 Tripwire，目的是建立一个工具，用于监视 UNIX 系统环境中的重要文件和目录发生的任何改变。

1997 年，G.H. Kim 和 W. Starnes 发起成立了 Tripwire 公司，目的之一是发布一个能够用于更多平台的 Tripwire 商业升级版本。2001 年 3 月，基于商业版的 Tripwire-2.x，Tripwire 公司发布了 Linux 操作系统环境下的开放源码的 Tripwire-2.3.1，该版本使用 GPL 作为许可证，对 Tripwire 的广泛应用起到了积极的推动作用。

8.6.1　Tripwire的原理与组成

Tripwire 对文件进行完整性检查的基本原理是：首先，根据完整性检查控制策略，为每个需要监控的文件生成一个指纹，并将它们存储到 Tripwire 数据库中；必要时，重新为每个需要监控的文件生成新的指纹，并将新指纹与 Tripwire 数据库中存储的指纹进行对比，据此，可以确定需要监控的文件是否已经被改动过。Tripwire 进行完整性检查

的基本原理如图 8.19 所示。

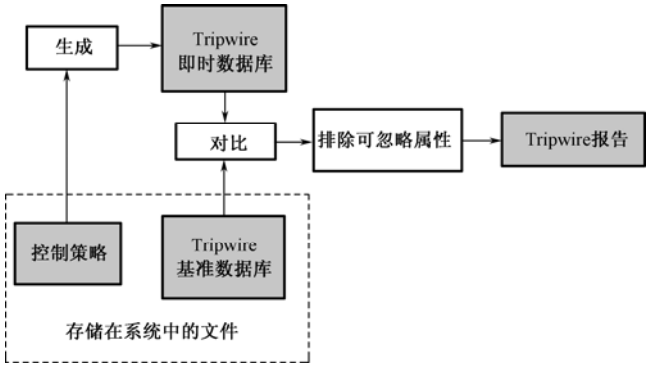


图 8.19　Tripwire 进行完整性检查的基本原理

Tripwire 是一款应用软件工具，在用户空间中运行。把 Tripwire 在进行完整性检查前预先生成的数据库称为基准数据库，把它在进行完整性检查时当场生成的数据库称为即时数据库。根据这两个数据库中的文件指纹的对比结果，可以生成相应的完整性检查报告。

Tripwire 使用单向消息摘要算法计算每个需要监控的文件的指纹信息，然后将其存储到基准数据库中。系统可以根据需要，随时计算需要监控的文件的指纹信息，生成即时数据库，并把即时数据库中的文件指纹与预先存储在基准数据库中的文件指纹进行对比，如果出现了差异，则向系统管理员发出报告信息。

消息摘要算法以任意的文件内容为输入，产生一个固定大小的输出结果（指纹）。Tripwire 支持的单向消息摘要算法包括：MD5、MD4、MD2、Snefru、HAVAL、SHA、CRC32 和 CRC16 等。其中，MD5、MD4 和 MD2 是 RSA 数据安全公司提出的算法，产生 128 位的结果，MD4 利用 32 位 RISC 架构；Snefru 是 Xerox 安全散列函数；HAVAL 是 128 位的消息摘要算法；SHA 是美国国家安全局设计的安全散列算法；CRC32 和 CRC16 分别是 32 位和 16 位的循环冗余校验码算法。

Tripwire 在控制策略的驱动下工作。控制策略和 Tripwire 数据库都是 Tripwire 赖以工作的重要信息。控制策略不仅描述了 Tripwire 应监控的文件和目录，还定义了用于鉴定违规行为的规则。一般情况下，对于/root、/bin 和/lib 目录及其中文件的修改可能是违规行为。除了控制策略和 Tripwire 数据库外，Tripwire 需要使用配置信息，用以描述 Tripwire 数据库、控制策略和 Tripwire 可执行程序在系统中的位置等。

早期的 Tripwire 版本使用一个配置文件（tw.config）描述控制策略和配置信息。现代的 Tripwire 版本使用策略文件（tw.pol）定义控制策略，使用配置文件（tw.cfg）描述配置信息。

为了防止被篡改，现代的 Tripwire 支持对其自身的一些重要文件进行加密和签名处理。这里涉及两个密钥：站点密钥（Site Key）和本机密钥（Local Key）。站点密钥用于保护策略文件和配置文件，如果一个站点中的多台机器具有相同的策略和配置，那么，

它们可以使用相同的站点密钥。本机密钥用于保护 Tripwire 数据库和 Tripwire 报告，不同的机器必须使用不同的本机密钥。

建立基准数据库是 Tripwire 在开始执行完整性检查任务前必须完成的工作。Tripwire 根据策略文件的描述确定需要监控的文件，然后，对文件系统中需要监控的文件进行扫描，生成 Tripwire 的基准数据库，主要过程如图 8.20 所示。

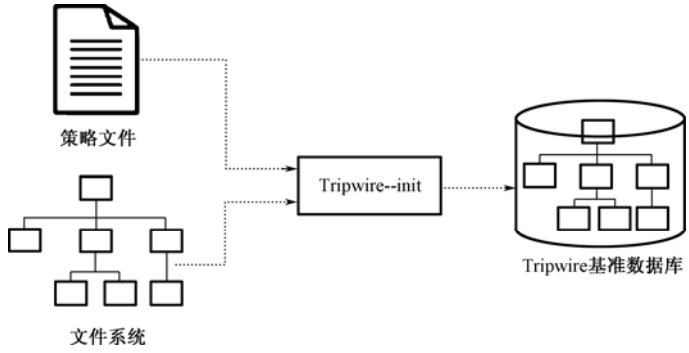


图 8.20　Tripwire 生成基准数据库

与生成基准数据库的方法一样，Tripwire 根据策略文件的描述，对文件系统中需要监控的文件进行扫描，生成即时数据库。通过对比文件在即时数据库与基准数据库中的指纹，可以生成 Tripwire 报告，完成对文件进行完整性检查的任务。Tripwire 进行完整性检查如图 8.21 所示。

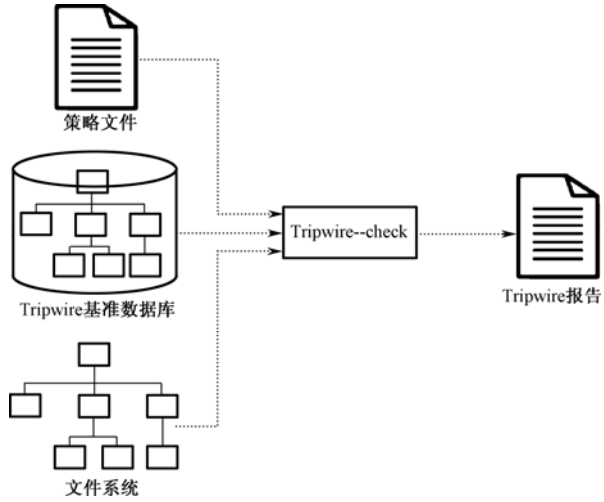


图 8.21　Tripwire 进行完整性检查

除了工作中涉及的策略文件、配置文件、Tripwire 数据库和 Tripwire 报告等数据型组成成分外，作为一个软件系统，Tripwire 包含 tripwire、siggen、twadmin 和 twprint 等程序型组成成分。其中，tripwire 是 Tripwire 完整性检查工具的主程序文件，siggen、

twadmin 和 twprint 是配套的实用程序。siggen 显示指定文件对应的指纹，twadmin 提供与 Tripwire 文件和配置选项相关的管理功能，twprint 以文本格式输出 Tripwire 数据库和报告文件。

8.6.2　Tripwire的工作模式

Tripwire 拥有四种工作模式：数据库初始化、完整性检查、数据库更新和交互式数据库更新。这四种工作模式是互不交叉的。

在数据库初始化模式下，Tripwire 根据策略文件的描述生成基准数据库，在基准数据库中，为策略文件中指定的每个文件生成一个记录，记录的内容包含文件名、索引节点属性、指纹信息、可忽略属性掩码、策略文件中对应的表项等。其中，可忽略属性掩码所列出的属性的变化可视为对文件的完整性没有影响。图 8.20 表示 Tripwire 在数据库初始化模式中工作。

在完整性检查模式下，Tripwire 根据策略文件的描述生成即时数据库，将即时数据库与基准数据库进行对比，生成新增文件和被删除文件的列表，并结合可忽略属性掩码生成文件被改动的报告。对于一个只是属性有变化的文件，如果变化的属性出现在可忽略属性中，则无须把该文件列入已被改动的文件之列，这就是图 8.19 中的“排除可忽略属性”的含义。图 8.21 表示 Tripwire 在完整性检查模式中工作。

只要文件系统中的文件发生了变化，就会出现文件系统的实际状态与基准数据库反映的状态不一致的现象。如果文件遭到了非法篡改，则需要恢复原始文件，以消除这种不一致性。如果是合法用户使文件发生了合理的变化，则应该更新基准数据库，以建立新的一致性。

Tripwire 可以在数据库更新模式或交互式数据库更新模式下更新基准数据库。在数据库更新模式下，Tripwire 根据系统管理员在命令行中给出的文件列表或策略文件表项，为指定文件生成 Tripwire 数据库记录，并最终生成新的基准数据库。在交互式数据库更新模式下，Tripwire 首先生成一个有关被改动的文件的列表，针对列表中的每一个表项，询问系统管理员是否确认要更新对应文件的信息，并根据问答结果更新基准数据库。

8.6.3　完整性检查策略的定义

看了前面两小节的介绍之后，知道 Tripwire 是怎样确定要执行的完整性检查任务的吗？相信你已经想到了，是的，它是按照控制策略的指示去执行任务的。

控制策略由策略文件进行定义，策略文件主要确定两件事情：一是解答要检查什么对象的完整性；二是解答如何检查那些对象的完整性。Tripwire 的策略文件中的规则可以把这两件事情联系起来，它的格式为

```
object_name -> property_mask;
```

其中，object_name 表示与检查有关的对象的路径名，property_mask 是属性标志，它指明对象中需要检查或可以忽略的属性。

Tripwire 的完整性检查中的对象可以是文件，也可以是目录。object_name 表示的路径名是绝对路径名，即从根目录开始的路径名。以下是几个常见的有效的对象路径名：/etc；/usr/lib；/etc/passwd；/etc/xinetd.conf；/bin/login。其中，前两个是目录的路径名，后三个是文件的路径名。

在属性标志中可以指明一组属性，每个属性由一个字符定义，每个属性字符前面可以带一个加号或减号，加号表示其后随的属性需要检查，减号表示其后随的属性可以忽略。如果字符 a 和 p 分别表示“访问操作的时间戳”和“权限位”属性，那么，以下属性标志表示要检查权限位的变化而忽略访问操作时间戳的变化：

```
+p-a
```

把对象的路径名和属性标志组合起来，便可定义所需要的规则，从而描述所需要的完整性检查控制策略。

例 8.22　试解释以下规则给 Tripwire 所确定的完整性检查任务：

```
/etc/xinetd.conf -> +p-a;
```

解答：该规则指示 Tripwire 检查文件/etc/xinetd.conf 的完整性，检查时，必须检查该文件的权限位变化情况，不必考虑该文件的访问时间戳的变化情况。如果文件的权限位发生了变化，则认为该文件的完整性受到了破坏。

以下这些是可以在 Tripwire 的策略文件中用于定义属性标志的属性字符及其含义。

（1）-：忽略后随属性；

（2）+：记录和检查后随属性；

（3）a：访问操作的时间戳；

（4）b：分配到的块数；

（5）c：索引节点的时间戳（创建/修改）；

（6）d：索引节点所在设备的 ID；

（7）g：属主的组 ID；

（8）i：索引节点号；

（9）l：文件大小不断增长（一个“增长中的文件”）；

（10）m：修改操作的时间戳；

（11）n：链接数（索引节点引用数）；

（12）p：权限位；

（13）r：索引节点指向的设备的 ID（仅对设备对象有效）；

（14）s：文件大小；

（15）t：文件类型；

（16）u：属主的用户 ID；

（17）C：CRC32 校验值；

（18）H：HAVAL 指纹；

（19）M：MD5 指纹；

（20）S：SHA 指纹。

例 8.22 给出的只是 Tripwire 中的一个简单的完整性检查策略定义规则，通过属性字符的灵活组合，可以定义出含义非常丰富的策略规则。

例 8.23　试解释 Tripwire 的以下策略规则所确定的完整性检查任务：

```
/usr/lib -> +pinugtsdbmCM-rlacSH;
```

解答：该规则指示 Tripwire 检查目录/usr/lib 的完整性。必须检查的属性有：权限位、索引节点号、链接数、属主的用户 ID、属主的组 ID、文件类型、文件大小、索引节点所在设备的 ID、分配到的块数、修改时间戳、CRC32 指纹、MD5 指纹。可以忽略的属性有：索引节点指向的设备的 ID、文件大小不断增长、访问时间戳、索引节点的时间戳（创建/修改）、SHA 指纹、HAVAL 指纹。

例 8.22 给出的规则中的属性标志比较长，通过采用给变量赋值的方法，可以把该规则表示成比较简捷的形式。

例 8.24　试采用变量赋值法把以下的 Tripwire 策略规则描述为一种比较简捷的形式：

```
/usr/lib -> +pinugtsdbmCM-rlacSH;
```

解答：以下形式描述的规则与题中给出的规则等价：

```
ReadOnly = +pinugtsdbmCM-rlacSH;
/usr/lib   ->   $(ReadOnly);
```

例 8.24 首先把常量形式的属性标志值赋给变量 ReadOnly，然后再利用变量 ReadOnly 的值定义完整性检查规则。这种方法能获得与普通程序设计语言中用变量代替常量所具有的相同效果。

实际上，ReadOnly 是 Tripwire 预定义的一个变量，无须重复定义，它表示一种常用的属性标志。例 8.24 中给出该变量的定义，仅仅是以它为例说明变量的使用方法而已。以下是 Tripwire 预定义的另外几个常用变量：

```
Dynamic = +pinugtd-srlbamcCMSH;
Growing = +pinugtdl-srbamcCMSH;
Device = +pugsdr-intlbamcCMSH;
IgnoreAll = -pinugtsdrlbamcCMSH;
IgnoreNone = +pinugtsdrbamcCMSH-l;
```

下面借助一个简单的例子考察通过 Tripwire 的策略规则定义 Tripwire 的策略文件的

基本方法。

例 8.25　以下是 Tripwire 的一个简单的策略文件的内容，其中的行号是为了说明问题的方便而添加的：

```
1.  # Critical configuration files
2.  (
3.  # rulename = "Critical configuration files",
4.  severity = 100
5.  )
6.
7.  {
8.  /etc -> $(ReadOnly) (rulename="/etc - critical config files");
9.  /etc/default -> $(ReadOnly) (rulename="/etc/default - critical config files");
10. /etc/inittab -> $(ReadOnly) (rulename="/etc/inittab - critical config files");
11. /etc/hosts -> $(ReadOnly) (rulename="/etc/hosts - critical config files");
12. /etc/xinetd.conf -> $(ReadOnly) (rulename="/etc/xinetd.conf - critical config files");
13. /etc/protocols -> $(ReadOnly) (rulename="/etc/protocols - critical config files");
14. /etc/services -> $(ReadOnly) (rulename="/etc/services - critical config files");
15. /etc/xinet.d -> $(ReadOnly) (rulename="/etc/xinet.d - critical config files");
16. }
```

第 1 行和第 3 行以#开头，属于注释。第 2～5 行定义的是一个规则属性，该规则属性对第 7～16 行中描述的所有规则有效，即所列规则的 severity 等级均定义为 100。

以第 8 行的规则为例，规则后面也定义了一个规则属性，该规则属性仅对第 8 行的规则有效，它通过 rulename 关键词给该行的规则定义了一个规则名，该规则名可以作为索引，用于在 Tripwire 报告文件中检索相应规则对应的结果信息。

上面以普通文本的格式对 Tripwire 的策略文件进行了分析，但实际上，Tripwire 中使用的策略文件（tw.pol）是经过加密和签名处理的，以非文本的格式存在。Tripwire 提供了非文本格式的策略文件（tw.pol）与文本格式的策略文件（twpol.txt）之间的转换方法。以下命令根据非文本格式的策略文件（tw.pol）生成文本格式的策略文件（twpol.txt）：

```
# twadmin --print-polfile > twpol.txt
```

其中的#表示命令由特权用户（系统管理员）执行。

以下命令根据文本格式的策略文件（twpol.txt）生成非文本格式的策略文件（tw.pol）：

```
# twadmin --create-polfile --cfgfile /etc/tripwire/tw.cfg \
        --site-keyfile site_key etc/tripwire/twpol.txt
```

同样，Tripwire 中使用的配置文件（tw.cfg）也是经过加密和签名处理的，以非文本的格式存在。Tripwire 也提供了非文本格式的配置文件（tw.cfg）与文本格式的配置

文件（twcfg.txt）之间的转换方法。以下命令根据非文本格式的配置文件（tw.cfg）生成文本格式的配置文件（twcfg.txt）：

```
# twadmin --print-cfgfile > twcfg.txt
```

以下命令根据文本格式的配置文件（twcfg.txt）生成非文本格式的配置文件（tw.cfg）：

```
# twadmin --create-polfile --cfgfile /etc/tripwire/tw.cfg \
        --site-keyfile site_key etc/tripwire/twpol.txt
```

文本格式是用户使用的需要，非文本格式是系统处理的需要。系统管理员以文本文件的形式编辑和定义 Tripwire 的策略和配置，并把它们转换成非文本格式的策略文件和配置文件，供 Tripwire 用于驱动文件完整性检查的行为。

8.6.4　Tripwire的基本用法

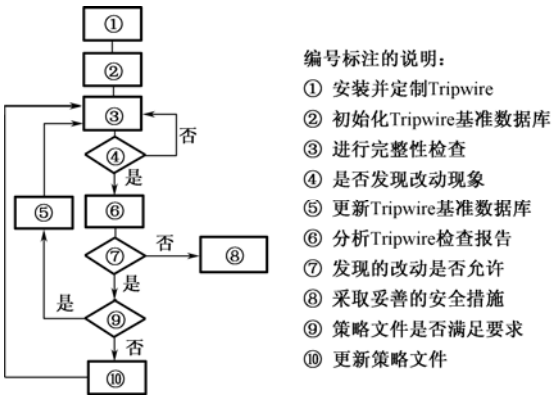


图 8.22　Tripwire 工具的应用方法

对于一个应用软件工具，在了解了它的核心思想和基本原理之后，分析一下如何把它用于解决实际问题。下面，综合前面三小节介绍的内容，以 Linux 操作系统环境为背景，简要地考察一下应用 Tripwire 工具检查文件系统中的文件的完整性的基本方法。Tripwire 工具的应用方法如图 8.22 所示。

应用 Tripwire 工具对文件系统中的文件进行完整性检查的关键是：

- 配置该工具的工作环境；
- 定义与用户需求一致的控制策略；
- 生成基准数据库；
- 周期性地进行完整性检查并分析检查报告；
- 必要时修改控制策略并更新基准数据库。

8.6.3 节着重讨论了基于策略文件的控制策略的定义方面的基本方法。有了完整性检查控制策略，可以使用下面的命令生成基准数据库：

```
# tripwire --init
```

使用下面的命令，就可以让 Tripwire 根据策略文件中的规则对文件系统中的文件进行完整性检查，并生成 Tripwire 报告：

```
# tripwire --check
```

以上这两个命令在图 8.20 和图 8.21 中也分别有所表示。

Tripwire 执行完整性检查操作后生成的 Tripwire 报告存放在由配置文件指定的地

方，配置文件给 REPORTFILE 变量所赋的值表示的就是 Tripwire 报告文件的路径名。通常，赋值方式是：

```
REPORTFILE = /var/lib/tripwire/report/$(HOSTNAME)-$(DATE).twr
```

其中，变量 HOSTNAME 的值是主机名，如 hostofswc，变量 DATE 的值是生成报告时的时间戳，形式为 20081021-112250（2008 年 10 月 21 日 11 时 22 分 50 秒），因此，报告文件路径名的形式是：

```
/var/lib/tripwire/report/hostofswc-20081021-112250.twr
```

至此，对应用 Tripwire 工具进行完整性检查需要考虑的常见问题的基本解决方法进行了扼要的介绍。需要注意的是：Tripwire 是进行周期性检查的，显然，在两次检查间隔内的修改对它来说是无从知晓的，这是它的适用场景。不过，任何工具都不可能解决所有的问题，Tripwire 当然也不例外，在实际工作中，明智的抉择是根据具体的情况选用合适的工具和解决方案。

8.7　本章小结

本章从完整性的经典模型、基于系统安全引导的完整性、基于安全协处理器的完整性、基于安全中央处理器的完整性、内核主导的完整性度量和文件系统的完整性检查等方面，讲解了系统完整性的度量与保护的基本思想、技术和方法。

本章介绍的完整性经典模型包括毕巴模型、克拉克-威尔逊模型和莫科尔树模型，前两者是完整性访问控制模型，后者是完整性度量模型。

毕巴模型用有序的完整性级别来刻画完整性，它规定：高完整性级别的主体可以写低完整性级别的客体，高完整性级别的主体可以执行低完整性级别的主体。毕巴低水标模型规定：主体 S 读客体 O 后，S 的级别取 S 与 O 的级别中的较小值。毕巴环模型对读操作不做控制。毕巴严格完整性模型规定低完整性级别的主体可以读高完整性级别的客体。

克拉克-威尔逊模型是一个面向事务的模型，它定义了良构事务、约束数据项、非约束数据项、完整性验证过程和转换过程等概念，规定只有转换过程才能对约束数据项进行操作。它从良构事务和职责分离两个方面去构造完整性控制框架，把系统中的数据项划分为约束数据项和非约束数据项，通过转换过程和完整性验证过程定义系统的行为，依靠证明规则和实施规则约束系统的行为，从而实现完整性控制。

莫科尔树模型是利用莫科尔树进行完整性度量的模型。莫科尔树是用于计算数据项的哈希值的二叉树，叶节点与数据项对应，内部节点与子节点的连接结果的哈希值对应。该模型在数据项分割的基础上实现数据项的完整性验证，它的基本出发点是依靠较小的内存空间开销实现较快的数据完整性验证，它要解决的问题是设计一个算法，以便能够快速验证任意一个数据项的完整性，并占用较少的内存空间。

AEGIS 安全引导模型实现了一种基于系统安全引导的完整性支持方法，它把系统组

件划分为 6 层，底层是基础 BIOS 和 AEGIS ROM，其中包含可信软件，其余各层的可执行代码在执行之前，由低层进行完整性检查。完整性检查通过哈希值和数字签名实现。该模型提供系统恢复机制，实现系统安全引导。

IBM 4758 模型实现了一种基于安全协处理器的完整性支持方法，IBM 4758 安全协处理器是一个物理上密封的缩微计算机，它插接在主机系统的 PCI 总线上，具有一定的独立性，能够运行独立的应用软件，其内部把组件划分为若干层，借助数字签名实现内部系统的安全引导。它提供了物理攻击反应、硬件锁访问控制、代码装入认证和消息认证码检查等多种功能，通过多种措施的综合运用来实现系统的完整性保护。

MIT 的 AEGIS 模型实现了一种基于安全中央处理器的完整性验证方法。它以进程的完整性为主要对象，以普通中央处理器为基础，在其中增加安全处理单元，从而提供对进程的完整性进行验证的功能。它采用的完整性验证方法是基于哈希值的方法，它从初始状态完整性支持、中断过程完整性支持、片上 Cache 的完整性支持、片外存储介质的完整性支持、结果信息的完整性支持等多个侧面，综合提供完整性支持。该模型通过硬件完整性验证单元验证进程在片外存储介质中的信息（包括代码和数据）的完整性，完整性验证单元实现了对莫科尔树模型的支持，按照莫科尔树模型的方法实施完整性验证。

IMA 模型实现了一种内核主导的完整性度量方法，它是一个基于 TPM 硬件芯片和 Linux 操作系统的模型，从两个方面进行完整性度量：一方面，度量基本可执行程序的完整性；另一方面，度量其他可执行内容的完整性和敏感数据文件的完整性。

IMA 模型主要从两个方面实现完整性的度量：其一是设计一个 measure 内核函数，用于对指定的内容进行完整性度量；其二是确定度量点，用于启动 measure 内核函数的完整性度量工作。measure 内核函数可以度量文件或内存区域的完整性。度量点设置在内核 load_module 例程、file_mmap 钩子函数和 sysfs 伪文件系统/sys/security/measure 中。该模型通过在内核空间和用户空间的度量点处启动内核主导的完整性度量操作，实现了整个系统范围的操作系统内核和用户空间进程的完整性度量。

Tripwire 是一个面向 UNIX 环境的著名工具，它实现了文件系统完整性检查的一种经典方法。Tripwire 的基本原理是：首先，根据完整性检查控制策略，计算文件的指纹信息，并将它们存储到基准数据库中；必要时，重新计算文件的指纹信息，生成即时数据库，并将即时数据库中的指纹与基准数据库中的指纹进行对比，据此判断文件是否受到了破坏。完整性检查控制策略由策略文件进行定义。策略文件主要确定两件事情：一是回答要检查什么对象的完整性；二是回答如何检查那些对象的完整性，策略文件中的规则可以把这两件事情联系起来。应用 Tripwire 工具对文件系统中的文件进行完整性检查的关键是：配置该工具的工作环境、定义与用户需求一致的控制策略、生成基准数据库、周期性地进行完整性检查并分析检查报告及必要时修改控制策略并更新基准数据库。



习题 8

1. 简要说明毕巴模型中的低水标模型、毕巴环模型及严格模型之间的主要区别。

2. 克-威模型的实施规则是从哪几个方面进行完整性访问控制的？它的证明规则是从哪几个方面确保完整性访问控制的有效性的？

3. 克-威模型是一个自主访问控制模型还是一个强制访问控制模型？为什么？

4. 设 U_i 是一个用户标识，TP_j 是一个转换过程，CDI_k 是一个受保护的数据项，请问克-威模型如何判断用户 U_i 是否可以通过执行 TP_j 来对数据项 CDI_k 进行操作？

5. 根据图 8.3 给出的莫科尔树，确定数据项 D_4 的完整性验证路径，说明路径中的数据项在验证数据项 D_4 的完整性时的用途，并以此为例说明定理 8.2 的含义。

6. 为了支持系统引导过程中的完整性验证，AEGIS 引导模型在普通计算机中增加了什么硬件组件？AEGIS 引导模型的可信根由哪些成分构成？

7. 对比图 8.5 和图 8.6，说明安全引导与可信引导的共性与区别。

8. 简要说明可以在系统引导过程中采用的基于哈希值和数字签名的完整性验证的基本方法。

9. 简要说明 IBM 4758 安全协处理器提供的基于硬件锁的访问控制的基本方法。

10. IBM 4758 安全协处理器主要采取哪些措施保护系统的完整性？各项措施分别实现什么目标？

11. MIT 的 AEGIS 模型在安全中央处理器中配备了哪些安全专用硬件单元？它们分别提供什么功能？

12. MIT 的 AEGIS 模型主要是通过哪些环节上的完整性措施来实现完整性支持的？各个环节上的措施分别实现什么目标？

13. 简要说明 MIT 的 AEGIS 完整性支持体系通过硬件单元实现莫科尔树模型的基本方法。

14. 简要说明内核主导的 IMA 模型进行完整性度量的基本方法。

15. 简要说明内核主导的 IMA 模型的完整性度量机制的实现方法。

16. Tripwire 文件完整性检查系统主要由哪些成分构成，结合这些成分，简要说明该系统的基本工作原理。

17. 在实际应用中，采用 Tripwire 系统进行文件完整性检查，用户需要完成哪些方面的主要工作，涉及哪些主要命令？



第三部分

拓　展　篇

第 9 章
基于主机的入侵检测



入侵检测技术是为保证计算机系统的安全而设计与配置的一种能够及时发现并报告系统中未授权或异常现象的技术。基于主机的入侵检测通常从主机的审计记录和日志文件中获得所需的主要数据源，并附加主机上的其他信息，如文件系统属性、进程状态等，在此基础上完成检测攻击行为的任务。本章首先简单介绍了入侵检测的基本概念，接着重点介绍基于主机的入侵检测，主要包括基于主机入侵检测技术的发展历程、基于主机入侵检测系统的信息获取和基于主机入侵检测的方法。

9.1　入侵检测概述

为了使读者对入侵检测的基本概念有一个清晰的了解，本节将介绍入侵检测的概念、系统模型、作用和分类等。

9.1.1　入侵检测的概念

计算机安全的 3 个基本目标是：机密性、完整性和可用性。安全的计算机系统应该实现这 3 个目标，保护自身的信息和资源。美国国家安全通信委员会下属的入侵检测小组（IDSG，Intrusion Detection System Group）在 1997 年给出关于“入侵”的定义：

定义 9.1　入侵是对信息系统的非授权访问及未经许可在信息系统中进行的操作。

攻击者之所以能够对计算机系统实施攻击，是因为计算机系统存在安全问题：操作系统的脆弱性；计算机网络的资源开放、信息共享及网络复杂性增大了系统的不安全性；数据库管理系统等应用系统设计中存在的安全性缺陷；缺乏有效的安全管理等。图 9.1 列出了常见的一些攻击手段。

下面介绍几个针对主机入侵的例子。

（1）口令破解，猜解弱口令是攻击者入侵主机系统行之有效的方法。从调查统计来看，常见的口令中 37%是自己姓名的中文拼音，23%是常用的英文单词，18%是计算机中经常出现的单词，7%是自己的出生日期，这些都是弱口令；只有 15%的口令属于良好口令。口令是主机安全的第一道防线，弱口令是口令安全的致命弱点。

（2）缓冲区溢出攻击，几十年来，缓冲区溢出一直引起许多严重的安全性问题。到目前为止，缓冲区溢出攻击仍是黑客常用的攻击手段之一。缓冲区溢出漏洞存在于各种操作系统、应用软件中。引起缓冲区溢出问题的根本原因是 C/C++没有数组和指针引用的边界检查功能，而开发人员往往忽略边界检查，一旦数组过长就造成了缓冲区溢出。缓冲区溢出攻击的目的在于触发某些具有特权运行的程序，使攻击者取得程序的控制权进而控制整台主机。攻击者往往攻击 root 程序，然后执行类似“exec(sh)”的执行代码来获得 root 权限的 shell。

（3）特洛伊木马攻击，特洛伊木马是一个包含在合法程序中的非法程序，是一种基

于远程控制的黑客工具。如果黑客获得了系统的存取控制权限，可以很容易地建立在此访问的后门。但如果他没有对系统的完全存取权限，往往通过诱使用户执行某些程序，而该程序捆绑有特洛伊木马程序，一旦用户执行了这些程序，木马就会留在目标系统中，用户在不知情的情况下运行的特洛伊木马程序可以直接侵入用户的计算机并进行破坏。黑客利用特洛伊木马程序可以肆意修改受害主机的配置、复制任何文件，窥视整个硬盘中的资料等，达到控制目标计算机的目的。

顾名思义，入侵检测便是对入侵行为的发觉。IDSG 在 1997 年给出的关于“入侵检测”的定义为：

定义 9.2　入侵检测是对企图入侵、正在进行的入侵或已经发生的入侵进行识别的过程。

所有能够执行入侵检测任务和功能的系统都可称为入侵检测系统（IDS，Intrusion Detection System），其中包括软件系统及软硬件结合的系统。

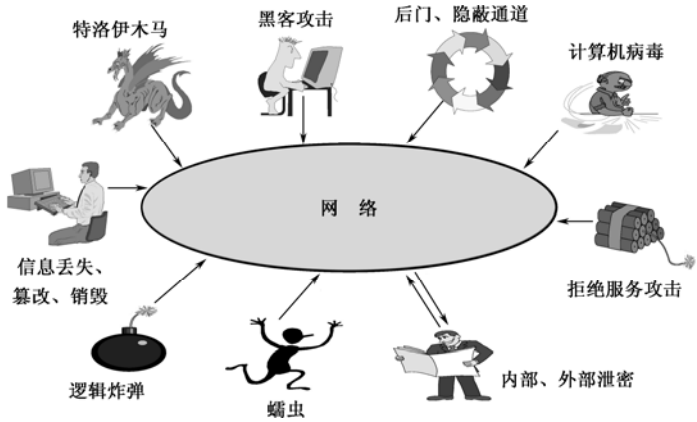


图 9.1　常见攻击手段

9.1.2　入侵检测系统模型

美国国防部高级研究计划局赞助研究了公共入侵检测框架（CIDF，Common Intrusion Detection Framework）。CIDF 阐述了一个入侵检测系统（IDS）的通用模型，CIDF 基本模型如图 9.2 所示。它将一个入侵检测系统分为以下组件：事件产生器（Event Generators）、事件分析器（Event Analyzers）、响应单元（Response Units）、事件数据库（Event Databases）。

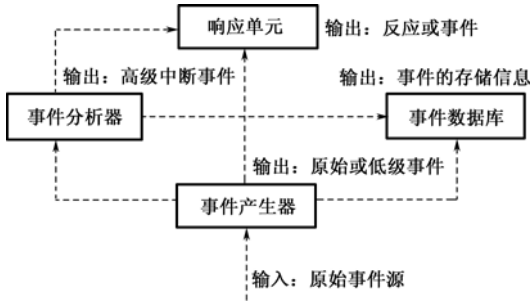


图 9.2　CIDF 基本模型

事件产生器的目的是从整个计算机环境中获得事件，并向系统的其他部分提供此事件。事件分析器分析得到数据，并产生分析结果。响应单元则对分析结果做出反应，它可以做出切断连接、改变文件属性等强烈反应，也可以只是简单报警。事件数据库是存放各种中间和最终数据的地方。

从以上的 CIDF 基本模型，可以总结出入侵检测的过程分为 3 步：信息收集、信息分析和结果处理。

（1）信息收集。入侵检测的第一步是信息收集，即从入侵检测系统的信息源中收集信息，收集内容包括系统、网络、数据及用户活动的状态和行为等。例如，系统日志文件、网络流量、非正常的目录和文件的改变、非正常的程序执行等。入侵检测在很大程度上依赖于所收集信息的可靠性和正确性。

（2）信息分析。经过第一步信息收集，收集到的信息量是非常庞大的，在这些海量信息中绝大部分信息都是正常状态的信息，而只有很少一部分信息可能表示入侵行为的发生。因此，要从大量的信息中找到异常的入侵行为，就需要对这些信息进行处理。可见信息分析是入侵检测的核心环节。信息分析的方法很多，如模式匹配、统计分析、完整性分析等。

（3）结果处理。当一个攻击企图或事件被检测到后，入侵检测系统按照预先定义的响应方式采取相应的措施。常见的响应方式有终止攻击、切断用户连接、禁止用户账号、记录事件的日志、向安全管理员发出提示性的电子邮件等。

9.1.3　入侵检测系统的作用

在计算机安全系统中，入侵检测系统的作用和功能如下：

- 监控、分析用户和系统的活动；
- 审计系统的配置和弱点；
- 评估关键系统和数据文件的完整性；
- 识别攻击的活动模式；
- 对异常活动进行统计分析；
- 对操作系统进行审计跟踪管理，识别违反政策的用户活动。

对于一个成功的 IDS 来讲，它不但可以使系统管理员时刻了解系统的任何变更，还能给系统的安全策略的制订提供指南。入侵检测作为安全技术，其主要目的有：

- 识别攻击者；
- 识别入侵行为；
- 检测和监视已成功的安全突破；
- 为对抗入侵及时提供重要信息，阻止事件的发生和事态的扩大。

可见，入侵检测是非常必要，它将有效弥补传统安全保护措施的不足。

9.1.4　入侵检测系统的分类

通过对现有的入侵检测系统和入侵检测技术的研究，可以从以下几个方面对入侵检测系统进行分类。

1．根据目标系统的类型分类

根据目标系统的类型不同，入侵检测系统通常可以分为基于主机的入侵检测系统（HIDS，Host-based Intrusion Detection System）、基于网络的入侵检测系统（NIDS，Network-based Intrusion Detection System）和混和入侵检测系统（Hybrid）。

1）基于主机的入侵检测系统

定义 9.3　基于主机的入侵检测系统检测的目标主要是主机系统和系统本地用户，检测原理是根据计算机操作系统的事件日志、应用程序的事件日志、系统调用、端口调用和安全审计记录发现可疑事件。

检测系统可以运行在被检测的主机或单独的主机上，系统结构如图 9.3 所示。入侵检测最早是从主机审计的基础上开始发展的，因而早期的入侵检测系统都是基于主机的入侵检测技术。

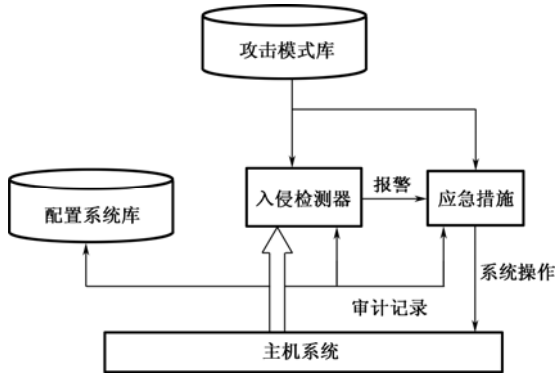


图 9.3　基于主机的 IDS 结构

基于主机的入侵检测的优点是：能够较准确地检测到发生在主机系统高层的复杂攻击行为，许多发生在应用进程级别的攻击行为是无法依赖基于网络的入侵检测来完成的；能确定攻击是否成功；监测特定主机系统活动；较适合有加密和网络交换器的环境；不需要另外添加设备。其缺点：可能因操作系统平台提供的日志信息格式不同，必须针对不同的操作系统安装不同的入侵检测系统；如果入侵者经其他系统漏洞入侵系统并取得管理员的权限，那将导致主机型入侵检测系统失去作用；可能会因拒绝服务攻击（DoS，Denail of Service）而失去作用；监控分析时可能会增加该台主机的系统资源负荷，影响被监测主机的效能。

2）基于网络的入侵检测系统

随着网络环境的普及，出现了大量基于网络的入侵检测系统。

定义 9.4　基于网络的入侵检测通过监听网络中的数据包来获得必要的数据来源，并通过协议分析、特征匹配、统计分析等手段发现当前的攻击行为。

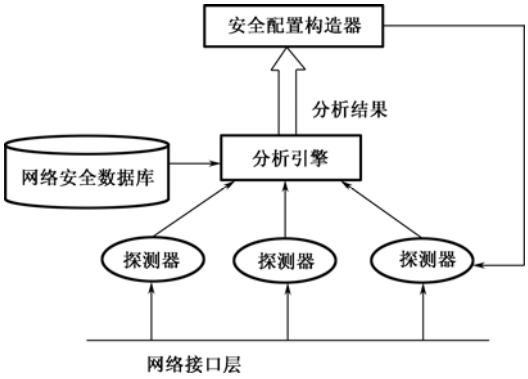


图 9.4　基于网络的 IDS 结构

网络型入侵检测系统担负着保护整个网段的任务。基于网络的 IDS 结构如图 9.4 所示。

探测器的功能是按一定的规则从网络上获取与安全事件相关的数据包，然后传递给分析引擎进行安全分析判断。分析引擎从探测器上接收到的数据包结合网络安全数据库进行分析，把分析的结果传递给配置构造器。配置构造器按分析引擎的结果构造出探测器所需要的配置规则。一旦检测到了攻击行为，NIDS 的响应模块就做出适当的响应，如报警、切断相关用户的网络连接等。

基于网络的入侵检测的优点是：成本低；可以检测到主机型检测系统检测不到的攻击行为；入侵者消除入侵证据困难；不影响操作系统的性能；架构网络型入侵检测系统简单。其缺点是：网络流速高时可能会丢失许多封包，容易让入侵者有机可乘；无法检测加密的封包；无法检测出直接对主机的入侵。

3）混和入侵检测系统

主机型入侵检测系统和网络型入侵检测系统都有各自的优缺点，混和入侵检测系统是基于主机和基于网络的入侵检测系统的结合，许多机构的网络安全解决方案都同时采用了基于主机和基于网络的两种入侵检测系统，因为这两种系统在很大程度上互补，两种技术结合能大幅提升网络和系统面对攻击和错误使用时的抵抗力，使安全实施更加有效。

2．根据检测方法分类

从数据分析的手段看，入侵检测通常分为两类：滥用（Misuse）入侵检测和异常（Anomaly）入侵检测。

定义 9.5　滥用检测（Misuse Detection）又称特征检测（Signature Detection）。假设入侵者活动可以用一种模式来表示，分析各种类型的攻击手段，并找出可能的“入侵特征”集合。利用这些特征集合或对应的规则集合，对当前的数据来源进行各种处理后，再进行特征匹配或规则匹配，如果发现满足条件的匹配，则指出发生了一次入侵行为。

滥用检测系统如图 9.5 所示。

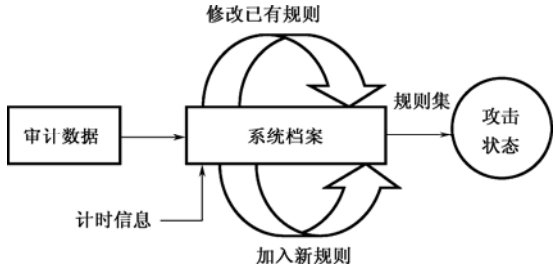


图 9.5　滥用检测系统

例 9.1　要在数据包中检查是否有“DEBUG”这样的关键字存在，如果有该字符串存在，则认为是某种攻击，描述如下：

```
get (data);
if (str == “DEBUG”)
    {
        进行入侵处理;
}
```

这种方法可以将已有的入侵方法检查出来，但对新的入侵方法无能为力。其难点在于如何设计模式既能够表达“入侵”现象又不会将正常的活动包含进来。

定义 9.6　异常入侵检测的假设条件是入侵者的活动异常于正常主体的活动，对入侵行为的检测可以通过观察当前活动与系统历史正常活动情况之间的差异来实现。

异常检测系统如图 9.6 所示。通常都会建立一个关于系统正常活动的“特征档案”并不断更新，然后将主体当前的活动情况与这个正常“特征档案”进行对比，当违反其统计规律时，认为该活动可能是“入侵”行为。

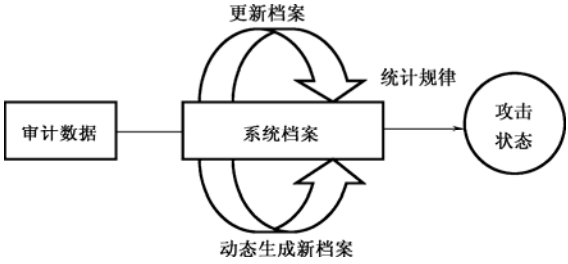


图 9.6　异常检测系统

例 9.2　假如对一般的员工来说，正常使用某台服务器的时间是 8:00～23:00，如果发现在凌晨 2:00 有人使用此服务器，那么就可以认为是异常情况，有可能是入侵行为或是非授权访问。此情况描述如下：

```
set normal-time = 8:00-23:00;
set state= normal;
get (login-time);
if (login-time != normal-time)
{
    set state= abnormal;
    进行异常处理;
}
```

异常检测的优点是：具有抽象系统正常行为从而检测系统异常行为的能力。这种能

力不受系统以前是否知道这种入侵的限制，所以能够检测新的入侵行为。大多数正常行为的模型使用一种矩阵的数学模型，矩阵的数量来自于系统的各种指标，如 CPU 使用率、内存使用率、登录的时间和次数、网络活动、文件的改动等。异常检测的缺点是：若入侵者了解到检测规律，就可以小心地避免系统指标的突变，而使用逐渐改变系统指标的方法逃避检测；另外，检测效率不高，检测时间比较长；最重要的是这是一种“事后”的检测，当检测到入侵行为时，破坏早已经发生了。

3．根据检测系统对入侵的响应方式分类

根据检测系统对入侵的响应方式分类，入侵检测可以分为：主动入侵检测系统和被动入侵检测系统。

（1）主动入侵检测系统在检测出入侵后，可自动地对目标系统中的漏洞采取修补、强制可疑用户退出系统及关闭相关服务等对策和响应措施。

（2）被动入侵检测系统在检测出对系统的入侵攻击后只是产生报警信息通知系统安全管理员，之后的处理工作由系统管理员来完成。

4．根据检测系统对入侵的响应时间分类

根据检测系统对入侵的响应时间分类，入侵检测可以分为：实时处理系统和非实时处理系统。

非实时的检测系统通常在事后收集的审计日志文件的基础上，进行离线分析处理，并找出可能的入侵行为踪迹，目的是进行系统配置的修补工作，防范以后的入侵。随着处理能力的提高，实时处理系统逐渐变得流行起来。主机检测系统实时监控审计日志的变化，并及时发现危险的入侵行为。

下面重点介绍基于主机的入侵检测系统。

9.2　基于主机入侵检测技术的发展历程及发展趋势

20 世纪 70 年代，随着计算机运行速度的加快、计算机数量的增加和体积的减小，对计算机安全的需求也显著增加。入侵检测技术自 20 世纪 80 年代早期提出以来，经过 20 多年的不断发展，已成为计算机安全防护领域内不可缺少的一种重要的安全防护技术。1990 年以前，大部分入侵检测系统都是基于主机的，它们对于活动性的检查限于操作系统审计数据及其他以主机为中心的信息源。本节将回顾基于主机的入侵检测技术在过去历史时期内的发展历程，加深读者对基于主机入侵检测技术的理解。

9.2.1　发展历程

1980 年，James Anderson 在给一个保密客户写的技术报告中提出了入侵尝试（Intrusion Attempt）和威胁（Threat）的概念，并将其定义为：潜在、有预谋的未经授

权访问信息、操作信息，致使系统不可靠或无法使用的企图。提出了改进安全审计系统的建议，以便用于检测计算机用户的非授权活动，并提出了基本的检测思路。

1986 年 Stanford Research Institute（SRI）的 Dorothy E. Denning 发表了经典论文“An Intrusion-Detection Model”，该文深入探讨了入侵检测技术，首次将入侵检测的概念作为一种计算机安全防御措施提出。提出了入侵检测的基本模型，并提出了几种用于入侵检测的统计分析模型。这篇文章后来被认为是入侵检测系统的开山之作。

1988 年 SRI 开始开发入侵检测专家系统（IDES，Intrusion Detection Expert System），它是一个实时入侵检测系统。采用了统计技术来进行异常检测，用专家系统的规则进行滥用检测。IDES 在实现双重分析（特征分析和异常检测）和实时分析两个方面迈出了关键的一步，该系统被认为是入侵检测研究中最有影响的一个系统。

1988 年，针对美国空军计算机系统的多用户环境，Los Alamos 国家实验室和 Haystack Laboratories 采用异常检测和基于特征（Signature）的检测，开发了 Haystack 入侵检测系统，来发现异常的用户活动。同时，出现了为美国国家计算机安全中心主机开发的多入侵检测及告警系统（MIDAS，Multics Intrusion Detection and Alerting System），该系统是在国家计算机安全中心的公共信息系统上应用的第一个入侵检测系统。

1989 年，Los Alamos 国家实验室的 Hank Vaccaro 为国家计算机安全中心和能源部开发了 W&S（Widson and Sence）系统，这是一个基于主机的异常检测系统。同年，Planning Research Corporation（PRC）公司开发了 ISOA（Information Security Officers Assistant），ISOA 后来用在了 PRC 入侵检测系统 PreCis 中。

从 1992—1995 年，在 IDES 的基础上、在以太网的环境下实现了产品化的入侵检测原型系统（NIDES，Next-Generation Intrusion Detection Expert System），它采用的方法更加通用、灵活，对目标系统和审计数据的类型没有限制。

以上的研究是检测对主机的入侵。这方面的研究仍未停止，2001 年 10 月 SRI 发布了 eXpert-BSM™ for Solaris Version 1.4，是目前比较先进的基于主机的 IDS。具有可升级、大量的事件分析能力、通用性能好及具有可插的组件等特点。

9.2.2　发展趋势

近年来，入侵技术无论从规模与方法上都发生了变化。入侵者在实施入侵或攻击时往往同时采取多种入侵手段，以保证入侵的成功率，并可在攻击实施的初期掩盖攻击或入侵的真实目的。针对这种情况基于主机的入侵检测技术需要有相应的发展。

（1）随着防火墙技术的发展，出现了基于主机的防火墙技术。因此，下一代基于主机的入侵检测系统需要和基于主机的防火墙技术集成，并加入病毒扫描、完整性检查等相关技术，这种混合的系统可以更有效地监控多种类型的攻击行为。

（2）另一个趋势是，基于主机入侵检测机制应向操作系统内核发展，目前的检测程序都是运行在操作系统内核之外的用户空间上的。将检测机制集成在内核本身将可以更

广泛、更有效地检测到可疑的攻击行为。

9.3　基于主机入侵检测系统的信息获取

基于主机的入侵检测系统用于检测分析的审计数据，一方面来自操作系统提供的信息，包括操作系统审计记录和系统日志；另一个重要来源是直接来自于应用程序，这些数据来自日志文件或其他类似的资源。下面分别介绍不同级别的审计数据的收集。

9.3.1　操作系统的审计记录

最早的入侵检测技术是采用操作系统的审计记录作为输入数据源的。

定义 9.7　操作系统的审计记录是由操作系统软件内部的专门审计子系统产生的，其目的是记录当前系统的活动信息，如用户进程所调用的系统调用类型及执行的命令行等，并将这些信息按照时间顺序组织成为一个或多个审计文件。

对于不同的操作系统类型，其审计记录的组织形式也会有较大的差异。因此，入侵检测系统必须考虑根据各种类型的操作系统的审计机制提供不同类型的数据收集模块。另外，还存在一个问题：操作系统审计机制设计和开发的初始目标并不是为了满足后来才出现的入侵检测技术的需求，因此常常会出现系统的审计记录无法提供所需的有用信息，或者是审计系统提供了过多无关的信息，因此，IDS 经常要访问操作系统直接获取所需信息。

尽管操作系统审计记录作为入侵检测输入数据源存在一些问题，但是操作系统的审计系统设计了保护机制，审计机制要求具有抗破坏性和不可旁路性，因此操作系统的审计记录的安全性和可信性得到了较好的保证。另外，操作系统审计记录提供了系统内核级的事件发生情况，反映系统底层的活动情况，如系统调用、文件系统修改和用户登录等，允许入侵检测系统识别所有用户的活动细节，为发现可能的入侵行为奠定了良好的基础。

不同操作系统的审计机制设计存在差异，主机活动的审计范围和类型也有不同。典型地，UNIX 系列操作系统（包括 Linux、Solaris、Sun OS 等）与 Windows 操作系统就存在较大区别，UNIX 系列操作系统不同类型之间的审计系统也存在若干差异。下面以两种流行的操作系统 Sun Solaris、Linux 和 Windows 为例介绍其审计机制。

1．Sun Solaris

Sun 公司的操作系统 Solaris 通过其基本安全模块（Basic Security Module，BSM）提供审计信息。BSM 是一个内核扩展模块，可以记录到系统调用层的日志事件（Log Event）。每一个 BSM 审计记录都表示一次审计事件（Audit Event）的发生。可以指定不同的审计层次，与安全相关的高层事件同样可以获得，如登录事件（Login Event）。

但对超级用户 root 没有审计记录，因此攻击者一旦得到超级用户的特权，则可以很容易地进行入侵活动。

BSM 的审计日志由一个或多个审计文件组成，每个审计文件包含多个审计记录。审计记录以二机制的形式进行存储。审计文件的内容可以利用 praudit tool 打印成人们可以读懂的形式。

例 9.3　BSM 审计记录分析。图 9.7 显示了由 praudit tool 打印 BSM 审计文件的记录。这些记录表示了系统调用 execve 的执行。每条记录的组成为：系统调用时间 → 用户 ID、有效用户 ID、真实用户 ID – 来源机器、系统调用 + 工作目录 + 命令行 + 系统调用返回结果。

```
Thu Aug 10 22:01:29 2004 -> UID:root EUID:root RUID:root - From machine:log1
execve() + /usr/bin/sparcv7/ps + cmdline:ps,-ef + success
Thu Aug 10 22:01:50 2004 -> UID:root EUID:root RUID:root - From machine:log1
execve() + /usr/bin/tail + cmdline:tail,/etc/system + success
Thu Aug 10 22:11:18 2004 -> UID:root EUID:root RUID:root - From machine:log1
execve() + /usr/bin/pwd + cmdline:pwd + success
Thu Aug 10 22:11:20 2004 -> UID:root EUID:root RUID:root - From machine:log1
execve() + /usr/bin/ls + cmdline:ls,-l + success
Thu Aug 10 22:11:33 2004 -> UID:root EUID:root RUID:root - From machine:log1
execve() + /usr/bin/ls + cmdline:ls,-l + success
```

图 9.7　BSM 审计记录

2．Linux

Linux 操作系统通过 SNARE（System Intrusion Analysis and Reporting Environment）提供内核层的审计信息。InterSect Alliance 发布了 SNARE，它是一个动态加载模块，来实现基于主机入侵检测系统和 Linux 的审计/事件日志。它支持审计记录的简单模式匹配，并利用图形化工具过滤收集的信息。SNARE 由三部分组成：内核动态加载模块 auditmodule、在用户空间运行的审计监控程序 auditd、图形界面的配置和报告工具 snare。内核动态加载模块 auditmodule 包装了一些比较危险的系统调用，如 execve、open、mkdir，它把这些系统调用放到一个信息收集的例程，收集进程和用户执行的一些有疑问的系统调用信息。接着，这个模块把获得的信息放到一个临时缓冲区，用户空间的监控程序 auditd 从这个缓冲区可以获得这些信息并进行处理。

为了便于普通用户理解，SNARE 提供一个用于日志分析的图形前端 snare。通过这个图形前端可以容易地进行事件日志参数的配置和事件的显示。图 9.8 就是一个 snare 图形化窗口。从这个屏幕快照中可以看到事件的类型、发生事件的时间和细节描述。

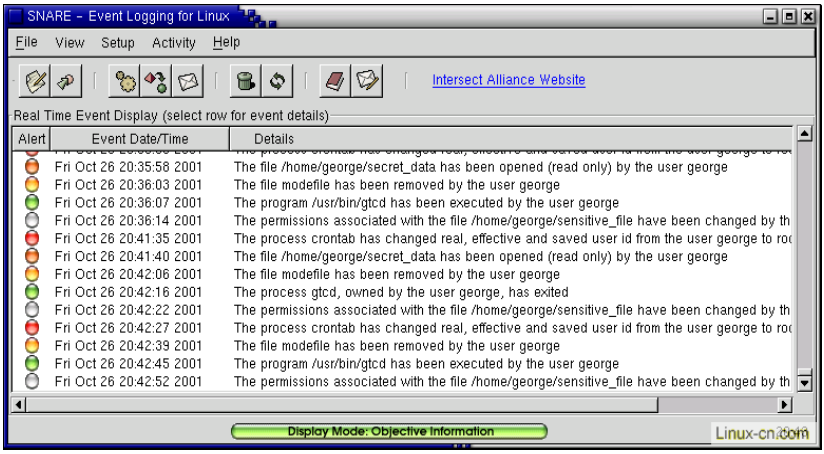


图 9.8　snare 图形化窗口

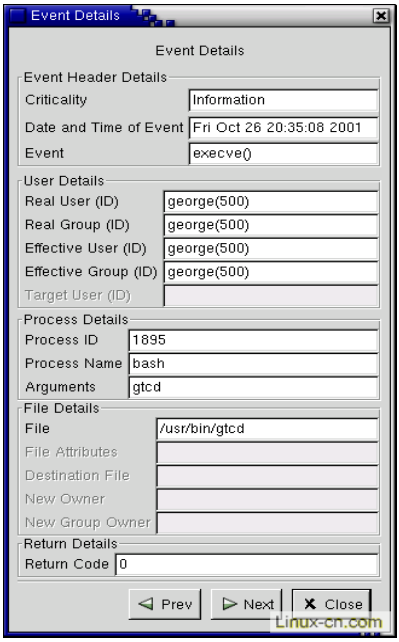


图 9.9　事件细节

完成审计配置后，SNARE 的图形前端的主窗口就会显示满足指定条件的审计事件。在主窗口中显示的是每个事件的概述。如果想知道某个事件的细节，可以使用鼠标双击这个事件，然后会弹出一个描述事件细节的窗口，如图 9.9 所示。

3．Windows

Microsoft Windows 操作系统也提供审计系统来实现基于主机的入侵检测。Windows XP 以事件日志机制提供数据源，记录和组织来自操作系统和其他数据源的信息。事件是用户进行的动作，可以根据事件的日志记录来识别和跟踪安全事件、资源使用情况、系统和应用程序中的错误。事件日志机制收集 3 种类型的系统事件：系统日志、安全日志和应用程序日志。

（1）系统日志（sysevent.evt）包括由 Windows 系统组件记录的事件，如启动时某个驱动程序加载失败；因为在机器的网络设定中要维护浏览器列表，在网络设置时，常常记录浏览器事件。系统事件由 Windows 自动配置，并记录在系统日志中。

（2）安全日志（secevent.evt）记录与安全性相关的事件，如合法或非法的登录企图，修改访问权限，系统启动、关闭，还包括与资源使用有关的事件，如创建、打开或删除文件。管理员可以指定将哪些事件记录在安全日志中。安全性事件根据审计策略被记录在安全日志中。注意：Windows 默认安装是不打开任何安全日志审核的，需要由管理员指定打开相应的审核。

（3）应用程序日志（appevent.evt）记载应用程序运行方面的事件。例如，数据库程序可以记录一次文件访问错误或应用程序的配置问题。事件的记录由开发者定义。当运行某些应用程序时，如 MS SQL Server 或 MS Exchange Server 日志的大小会变得很大。

这三种日志都可以通过事件查看器 Windows Event Viewer 进行查看和管理，Windows XP 事件查看器如图 9.10 所示。

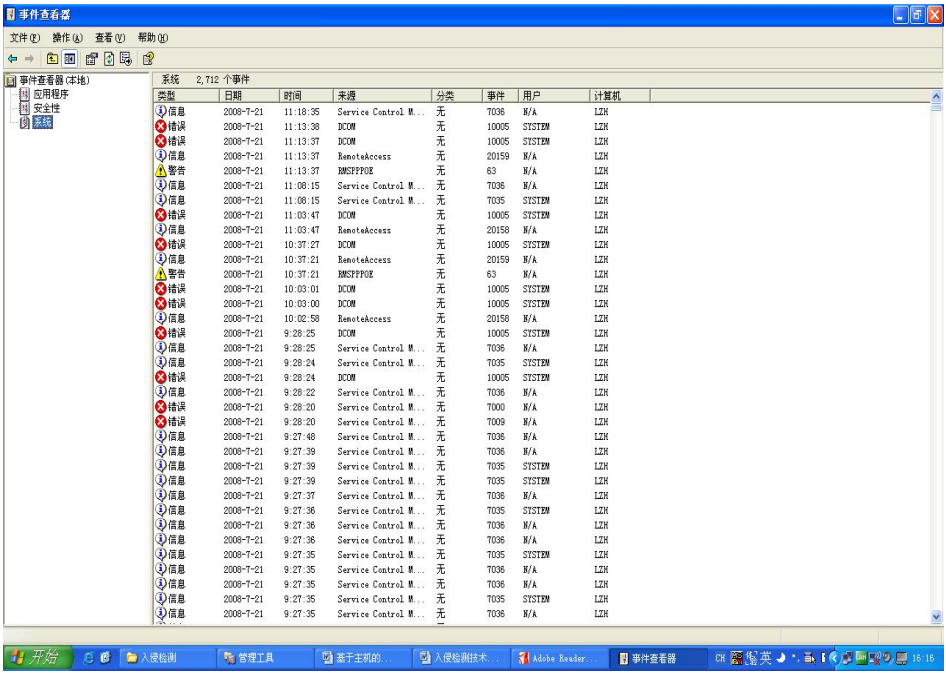


图 9.10　Windows XP 事件查看器

事件查看器中 3 种类型日志有统一的格式如下。

（1）类型：有五种事件类型，错误、警告、信息、成功审核、失败审核。

（2）日期：事件产生的详细日期。

（3）时间：事件产生的详细时间，精确到秒。

（4）来源：事件的产生者。

（5）分类：对事件分类，如系统事件、特权使用、登录/注销等。

（6）事件：事件 ID。用不同的 ID 来表明唯一的事件。

（7）用户：用户名称，包括 N/A、system、Administrator 等。

（8）计算机：计算机名称。

利用事件查看器可以根据需求快速地查看和筛选事件日志，如根据事件来源、事件类型、事件 ID、计算机名称等。事件日志可以保存为多种形式：如果以日志文件格式存档，则可以在事件查看器中重新打开它；如果存档为事件日志文件（*.evt），将保留所有事件记录的二进制数据；如果存档为文本或逗号分隔的格式（分别为*.txt 和 *.csv），

则可以在文字处理或电子表格之类程序中重新打开。通过查看和分析事件日志，可以检测出需要系统管理员重视的问题，如安全问题、系统问题、对资源进行重新分配等。

9.3.2　系统日志

除了审计记录，还可以获得审计和安全信息的地方是操作系统的日志文件。几乎所有的操作系统都拥有自己的日志机制。系统使用日志机制记录下主机上发生的事情，对追踪攻击者痕迹非常关键。

UNIX 操作系统的主要日志文件可以分成 3 类。

（1）二进制登录日志文件，包括 utmp、wtmp、lastlog，使系统管理员可以跟踪谁在何时登录到系统。有关当前登录用户的信息记录在文件 utmp 中；登录进入和退出记录在文件 wtmp 中；最后一次登录记录在文件 lastlog 中。utmp、wtmp、lastlog 是二进制文件，不能用普通文件查看器查看，只能通过系统命令如 who、w、users、finger、last 等来查看，内容主要包括：用户名、终端、登录时间、登录 IP、CPU 使用时间、正在运行的进程、退出时间等。

（2）第二类是进程日志，由系统内核生成。当一个进程终止时，系统内核为每个进程在进程日志文件（pacct 或 acct）中写入一条记录。能记录用户使用的每条命令，对于跟踪系统问题十分有用。例如，想知道谁在什么时间访问了某个重要文件，通过进程日志就可以得到相关信息。进程日志默认情况下并不激活，它必须显式地启动。一旦被激活，就可以使用 lastcomm 命令监测系统中任何时候执行的命令。记录的所有命令的有关信息包括：命令名、用户、终端、命令占用的 CPU 时间和时间戳等。

（3）第三类是 syslogd 日志，由 syslogd 生成并维护。各种系统守护进程、内核、模块使用 syslogd 记录下自己发出的消息。syslog 可以记录系统事件，可以写消息到一个文件或设备，或者直接给用户发送信息。通过 syslogd 生成的文件有如下统一的格式：时间戳、主机名、消息发送者、消息描述。

例 9.4　syslogd 生成文件中的一行：

Jun 12 11:06:11 Invent passwd [337]: password for ‘progs ’ changed by ‘root’

表示：名为 Invent 的主机收到来自 passwd 的消息，描述用户 progs 的口令被 root 改变了。

例 9.5　下面是从 Red Hat 8.0 系统的“message”日志中选出的几行记录：

Jan 5 18:20:39　jeep su (pam_unix) [32478]: session opened for user bob by (uid = 0)

Jan 5 18:20:47　jeep su (pam_unix) [32516]: authentication failure;

Logname=uid=502 euid=0 tty=ruser=bob rhost=user=root

Jan 5 18:20:53　jeep su (pam_unix) [32517]: authentication failure;

Logname=uid=502 euid=0 tty=ruser=bob rhost=user=root

```
Jan 5 18:21:06   jeep su (pam_unix) [32519]: authentication failure;
    Logname=uid=502 euid=0 tty=ruser=bob rhost=user=root
```

从第一行可以看到一个叫“bob”的用户打开了一个会话，这通常表示一个拥有“bob”账号的用户登录系统。从接下来的几行可以看到“bob”试图成为超级用户“root”时验证失败。

但是，syslog 系统日志有一些缺点。例如，它以任意的格式记录文本信息，因此自动分析系统日志输出信息就很困难。另外，系统日志工具使用多个日志文件来分类各种事件，分类方法是任意的、静态的，会造成相关联的事件被分在不同的日志文件中，结果使重要的上下文信息丢失，最后，提供的告警和响应机制受到限制。

9.3.3　应用程序的日志文件

前面讨论了操作系统日志文件，现在来介绍应用程序特殊的日志文件。特别感兴趣的是错误日志，因为，恶意行为经常会导致一些副作用从而被应用程序的错误检查程序检测到。

应用程序的日志文件中包含了非常细节的信息，这是它的优点。但是也存在缺点，不同程序的文件格式有很大差异，因此入侵检测系统需要去适应各个不同的应用程序。另一个缺点是日志信息是按时间顺序记录的，应用程序出现的问题执行完以后才记录到日志文件中，因此，入侵检测系统不能事先起作用。

例 9.6　应用程序 FTP 日志的例子。

FTP 日志默认情况为每天生成一个日志文件，包含了该日的一切记录。如 ex021225 就是 2002 年 12 月 25 日产生的日志，如下：

```
# Software: Microsoft Internet Information Services 5.0 （微软 IIS 5.0）
# Version: 1.0    （版本 1.0）
# Date: 20021225 04:11:55 （服务器启动时间）
04:11:55 127.0.0.1 [1] USER administrator 331 （IP 地址为 127.0.0.1 用户名为 administrator 的用户试图登录）
04:11:58 127.0.0.1 [1] PASS-530 （登录失败）
04:12:04 127.0.0.1 [1] USER nt 331  （IP 地址为 127.0.0.1 用户名为 nt 的用户试图登录）
04:12:06 127.0.0.1 [1] PASS-530 （登录失败）
04:12:09 127.0.0.1 [1] USER cyz 331 （IP 地址为 127.0.0.1 用户名为 cyz 的用户试图登录）
04:12:12 127.0.0.1 [1] PASS-530 （登录失败）
04:12:32 127.0.0.1 [1] USER administrator 331 （IP 地址为 127.0.0.1 用户名为 administrator 的用户试图登录）
04:12:34 127.0.0.1 [1] PASS-230 （登录成功）
04:12:41 127.0.0.1 [1] MKD nt 550          （新建目录失败）
04:12:45 127.0.0.1 [1] QUIT-550 （退出 FTP 程序）
```

从上面的日志可以看出 IP 地址为 127.0.0.1 的用户一直试图登录系统，换了 4 次用户名和密码才成功。例 9.6 中入侵者最终是用 administrator 用户名进入的，那么就要考虑更换此用户名的密码，或者重命名 administrator 用户。

例 9.7　Web 日志文件，如下所示：

10.0.1.2 - - [02/Jun/1999:13:41:37 -0700] "GET /a.html HTTP/1.0" 404 194

这条日志记录描述了来自 IP 地址为 10.0.1.2 主机请求访问文件 a.html，但该文件不存在，Web 服务器返回一个响应包含 194 个字节。

这条日志记录并没有包括 HIDS 需要的所有信息，如头信息是否太长或畸形、请求过程用了多长时间、服务器如何分析这个请求、这个请求转化为哪个本地文件等。

有一些应用程序，可以定制日志信息包含更多的信息，以便分析。但是，在大多数系统中，并不是解释每个操作需要的所有内部信息都记录在日志中，如果激活所有日志工具，就会耗尽日志存储空间或加速系统性能降低。

为了消除应用程序审计数据收集中日志文件存在的局限性，提出了一些 HIDS 与应用程序集成的系统，在这样的系统中，HIDS 监测应用程序内部的工作，在应用程序解释它的同时分析相关数据。这样就有机会在恶意操作执行之前检测到它，并有可能阻止其执行。HIDS 与应用程序紧密集成可以获得更多的信息（如一些不写入日志文件的本地信息），不需要去推测恶意操作的解释和结果，误报减少。例如，Web 服务器中的一个模块可以观察到全部 HTTP 请求，包括头信息。不用分析 Web 服务器的配置文件，该模块能确定 HTTP 请求映射到本地文件系统的哪个文件，它还能确定这个文件是否作为一个 CGI 程序来处理（CGI 程序既不能在网络传输数据中看到，也不能在日志文件中看到）。为了成功地将一个 HIDS 集成到一个应用程序中，应用程序必须提供适当的接口。

应用程序中集成审计数据收集模块的思想应用于蜜罐中。美国学者 L．Spizner 是一个著名的蜜罐技术专家。2004 年，他对蜜罐做了这样的一个定义：蜜罐是一种资源，它的价值是被攻击或攻陷。这就意味着蜜罐是用来被探测、被攻击甚至最后被攻陷的。通常，蜜罐是一个主机或网络用已知漏洞有意配置的，这些漏洞故意暴露在公共网络段，用以引诱入侵企图。蜜罐系统最为重要的功能是对系统中所有操作和行为进行监视和记录，可以学习攻击者的行为，另外，它还可以延缓和转移攻击者对更有价值的目标的攻击。

9.4　基于主机的滥用入侵检测方法

在基于主机的入侵检测系统中，常使用滥用入侵检测方法。滥用入侵检测系统中存有大量的攻击行为描述（或攻击特征），通过与获得的审计数据进行特征匹配来发现攻击行为。这里所说的“特征匹配”的实现手段各不相同，有最基本的字符串匹配，还有基于状态转移的分析模型等。

9.4.1　操作系统层的滥用入侵检测方法

滥用入侵检测系统通常使用攻击语言（attack language）来描述检测到的攻击行为。这些语言提供了提取攻击行为特征的机制，比较著名的有斯坦福研究所（SRI）的 EMERALD 入侵检测系统中利用 PBEST（Production-Based Expert System Toolset）进行入侵特征推理，美国加州大学圣巴巴拉分校（UCSB）提出并实现的 STAT（State Transition Analysis Tool）采用 STATL 来描述攻击场景。

STAT 在 UNIX 环境下的具体实现又称为 USTAT（STAT for UNIX）。在 USTAT 中用 STATL 描述每一种攻击过程所代表的状态转移图。下面以 STATL 为例详细介绍如何用语言来描述复杂的入侵行为。STATL 将攻击者的入侵行为描绘为一系列状态（State）和转移（Transition）的组合。状态用来描述在攻击过程中系统不同时间点上的特性。显然，描述系统的全部状态是不可行的，因此，STATL 使用变量来记录那些最能代表攻击行为特征（如计数器的值、文件的所有者）的系统状态。每一次状态转移都有相关联的操作（Action），这个操作是引起状态转移的特定事件。操作的例子如打开一个文件或执行一段应用程序。与操作相关的事件空间受转移断言（Transition Assertion）的制约，转移断言是与操作相匹配事件的过滤条件。例如，一组断言要求一个文件以指定的模式打开（如只读模式）或要求一个应用程序按照预先定义的安全规则执行。对于每个状态节点，都对应着一组状态断言，当满足该组断言后，本次从上个状态到本状态的转移过程才成功发生。

同一个攻击行为可能在同一时间激活若干个事件。因此，同样的场景原形有一组实例，这个场景原形是表示场景的定义和全部环境，场景实例表示当前进行的各个攻击行为。

一个场景中实例的演变由场景定义的转移操作类型决定。一个转移操作可以是消耗的（consuming）、非消耗的（nonconsuming）或延展的（unwinding）。一个非消耗的转移操作用来表示发生攻击的一个步骤，这一步可以使攻击行为从初始状态向更进一步的状态转变。因此，当一个非消耗的转移操作发生时，原始的状态保留有效，目的状态也变成有效。例如，假如一个攻击行为分为两步，第一步：攻击者对 SUID shell 脚本创建一个链接（Link）文件。第二步：攻击者通过已创建的链接执行该脚本。因此，第二步使以前的状态仍然有效，也就是说，通过相同的链接文件可以触发该脚本的另一个执行。从语义上说，触发一个非消耗的转移操作则创建一个新的场景实例，而原始的实例仍在原始的状态，新的实例在转移触发的目标状态。相对应地，触发一个消耗的转移操作将使一个特定攻击行为的原始状态无效，从语义上说，触发一个消耗的转移操作不能产生一个新的场景实例，它仅改变了原始的状态。延展的转移操作表示了反转的情况，它用来描述这样的事件和情形，当一个或多个场景实例向前推进无效时，需要回到这之前的状态。例如，删除了一个文件致使完成一个攻击需要的条件无效，因此，相应的场

景实例要返回到之前的一个状态，如该文件被创建前的状态。

例 9.8　以 ftp-write 攻击为例说明状态转移分析。

攻击者用 ftp 服务创建一个伪 .rhosts 文件，存放在完全可写的 ftp 根目录下。利用这个已创建的文件，攻击者利用 rlogin 服务不需要提供密码便可以打开一个远程会话。推而广之，这种攻击可以使攻击者在任何一个用户的根目录上创建一个伪 .rhosts 文件，利用这个已建好的文件不需密码便可登录。对于这种普遍意义的 ftp-write 攻击可以用状态转移图来表示，如图 9.11 所示。状态转移图由两个基本的组件构成：表示系统状态的节点和表示特征转移操作的弧线。

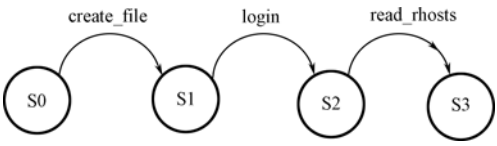


图 9.11　ftp-write 攻击状态转移图

图 9.11 中不同类型的箭头代表不同的转移操作类型：实心弧线有一个箭头表示非消耗的转移操作，实心弧线有两个箭头表示消耗的转移操作。

下面给出用 STATL 表示的 ftp-write 攻击过程。

```
use bsm, unix;
scenario ftp_write
{
int user;
int pid;
int inode;
initial state s0 { }
transition create_file (s0 -> s1)
nonconsuming
{
[WRITE w] : (w.euid != 0) && (w.owner != w.ruid)
{ inode = w.inode; }
}
state s1 { }
transition login (s1 -> s2)
nonconsuming
{
[EXECUTE e] : match_name(e.objname, "login")
{
user = e.ruid;
pid = e.pid;
}
}
```

```
state s2 { }
transition read_rhosts (s2 -> s3)
consuming
{
[READ r] : (r.pid == pid) && (r.inode == inode)
}
state s3
{
{
string username;
userid2name(user, username);
log("remote user %s gained local access", username);
}
}
}
```

关键字 Scenario 定义了一个攻击场景 ftp_write，其中包括四个状态定义 s0、s1、s2、s3，及三个转移定义：transition create_file（从状态 s0 向状态 s1 转移）、transition login（从状态 s1 向状态 s2 转移）、transition read_rhosts（从状态 s2 向状态 s3 转移）。STATL 中规定必须定义一个初始状态，initial state s0 { }，这里是 s0。在转移 transition create_file 中定义了一个特征操作[WRITE w]，在转移 transition login 中定义了一个特征操作[EXECUTE e]，在转移 transition read_rhosts 中定义了一个特征操作[READ r]。

这个场景检测到这样的事件序列，一个不具有超级用户权限的用户，对一个他没有权限目录下的文件进行了创建或写的操作，并运行登录程序，然后去读可疑的文件。从 BSM 事件类型中提取 WRITE、EXECUTE 和 READ 事件。match_name() 和 userid2name()完成匹配和用户 ID 转换的功能。

在滥用入侵检测系统中，STAT 基于状态转移分析的入侵检测技术，将攻击者的入侵行为描绘为一系列的特征操作及其所引起的一系列系统状态转移过程，在状态转移图和目标系统审计记录的基础上，不断跟踪攻击者在完成整个攻击过程中所必须完成的每个关键步骤。因此可以快速直观地检测到入侵行为。

9.4.2　应用程序层的滥用入侵检测方法

应用程序层的滥用入侵检测是通过监控应用程序的日志文件来检测入侵行为的。一个最简单的用来监控应用程序审计数据的系统是 Swatch（Simple Watch）。它是基于特征的系统，通过 UNIX 的 syslog()工具将监控信息写入日志文件。它的想法是要防止系统管理员受困于大量日志数据。这个工具监控日志文件，滤掉不想要的数据。基于日志中

的模式获取一个或更多用户指定的行为。Swatch 能监控添加到日志文件中的信息，当发生严重系统问题时，Swatch 能向系统管理员报警。

另一个监控日志文件的工具是 logSTAT，它还可以描述更复杂的攻击场景。使用前面介绍过的 STATL，logSTAT 将状态转换分析技术应用于系统日志文件。一个与 logSTAT 相关的工具 WebSTAT 将状态转换分析应用于特定的应用程序日志文件。Apache 创建 Web 服务器日志文件，已经实现的一些攻击场景包括：恶意 Web 爬虫场景、重复的失败访问场景和缓冲区溢出检测场景等。

例 9.9　Cookie 盗用攻击场景。

Cookies 就是服务器暂时存放在用户的计算机里的资料（.txt 格式的文本文件），好让服务器用来辨认用户的计算机。当用户在浏览网站的时候，Web 服务器会先传送一份资料放在用户的计算机上，Cookies 会帮用户记录在网站上所打的文字或一些选择。当下次用户再访问同一个网站时，Web 服务器会先看看有没有上次留下的 Cookies 资料，有的话，就会依据 Cookie 里的内容来判断使用者，送出特定的网页内容给用户。因此，Cookie 是一种 HTTP 管理机制，Web 应用程序开发者通常用它来跟踪会话。

Cookie 盗用场景可以检测到 Cookie 作为会话 ID 被多个用户非正当利用。这种情况通常表示恶意用户企图截取合法用户的会话，在没有授权的情况下访问受保护的 Web 资源。一个远程用户映射 Cookie 到一个 IP 地址，该场景开始记录 Cookie 会话的发布，每个用户使用一个 Cookie 是合理的，当检测到两个不同的用户使用同一个 Cookie 时，则认为是恶意行为。如果一个用户使用了另一个用户的合法 Cookie 会话，则预示着一次攻击开始了，要发出报警。

9.5　基于主机的异常入侵检测方法

在异常入侵检测系统中，用得最早也是最多的方法是统计分析方法。异常入侵检测技术是对滥用入侵检测技术的补充。异常入侵检测是基于用户和应用程序的正常行为模式，称为特征档案（Profile）。Profile 是用一组测量值对特定主体正常行为进行的描述。所定义的主体类型有：单个的用户、用户组、主机和整个系统。任何与 Profile 相违背的行为都解释为攻击。通过定义期望的正常行为就可以检测到任何异常行为，这些异常行为可能是入侵也可能不是。异常入侵检测的主要优点是可以检测到未知的攻击行为，但同时会造成较高的误报率，即合法的事件被判为是恶意的。

9.5.1　操作系统层的异常入侵检测方法

在操作系统层，以系统调用为审计数据进行程序行为的异常检测。系统调用是用户程序和系统内核的程序接口，用户程序可以通过系统调用陷入到操作系统内核执行相关

的功能和操作。用户程序最常用的 fork()、open()、read()、write()、exit()、execve() 等函数和操作实际上都是通过系统调用来执行的。因此，任何一个应用程序每一次运行的生命周期中都将形成一个以时间先后为序的系统调用序列，而应用程序根据其运行参数、运行环境的不同，形成的系统调用序列是不同的。入侵者利用系统漏洞或程序漏洞改变当前用户的可执行范围，运行恶意程序，达到入侵的目的。与此同时，程序的执行路径发生了质的改变，表现为系统调用序列与正常执行应用程序的序列间存在较大的差异，形成较大的波动。在 1996 年，Forrest 等人首先提出一个基于系统调用的简单有效的检测方法。每一个程序可用由它执行时产生的迹（Trace）来刻画，迹是由该程序在从开始运行到结束的过程中产生的系统调用组成的有序序列。例如，某个程序产生的迹为

R=(open，read，mmap，mmap, …，open，getrlimit，mmap，close)

在该“迹”中，系统调用的参数已被滤除。他们的研究表明程序的正常行为可以由在它的迹中出现的局部系统调用序列集来刻画，并且可以根据偏离这些局部序列集的程度来确定正在运行的程序违反安全策略的程度。

因此，对于异常行为的检测，必须先建立合法系统调用序列模型，该模型通过分析正常程序执行时的系统调用来建立。也就是说，通过观察系统调用来跟踪合法程序运行情况，“学习”正常行为。该类系统中一个例子是：在学习阶段（也称为训练阶段），收集某些特定的指定长度的系统调用序列；在检测阶段，所有当前的系统调用与已建立的合法系统调用集合进行比较，如果不匹配则发出一次报警。

例 9.10　选定系统调用序列的检测长度为 k。例如（k=3），假设观测到程序正常执行的系统调用命令（不包括命令参数）为

mmpa，rnrnap，open，read，mmap，mmap，open

则以上调用产生的调用序列为

{mmap，mmap，open}
{mmap，open，read}
{open，read，mmap}
{read，mmap，mmap}

将以上序列存入正常行为模式数据库中，现在假设在系统的入侵检测阶段截获的调用序列为

open，mmap，read，mmap，open，read，mmap

同样选定调用序列长度 k=3，那么被检测序列所产生的系统调用序列为

{open，mmap，read}
{mmap，read，mmap}
{read，mmap，open}
{mmap，open，read}
{open，read，mmap}

其中有 3 条不匹配的记录，不匹配概率为 60%，如果选定报警阈值为 40%，则该观测到的序列将导致监控进程通过用户界面向管理员报警。

另一类基于系统调用的异常入侵检测系统，关注分析系统调用参数而不是系统调用序列。检测的输入数据集合由操作系统记录的系统调用顺序流 $S=\{s_1, s_2, \cdots\}$组成，每一个系统调用 $s \in S$ 有一个输出值 r^s 和一些参数值列表$<a^s_1, \cdots, a^s_n>$。每个系统调用建立一个特定的档案（Profile），这个 Profile 由正常系统调用对应的正常参数值表示。通过评估系统调用的各个参数特征，确定各个正常的参数值，如字符串长度。模型有“学习”和“检测”两种工作状态。在“学习”状态时，通过观测示例，描述正常行为来训练模型。在“检测”状态时，输出系统调用参数值发生的概率，它表示了已建立的 Profile 中某个特征值的可能性。特征值在足够低的概率时就预示着一个潜在的攻击。

例 9.11　字符串长度模型。

通常，系统调用字符串参数表示规范的文件名，指向一个文件系统入口。当文件被访问（open,stat）或执行（execve）时，这些参数会被用到。它们的长度通常不会超过 100 个字符，并且基本上由人们能读懂的字符组成。字符串长度模型的目的是通过估计一个字符串参数长度的分布情况检测那些背离正常情况的行为。因此，可以检测到格式化字符串攻击等。

所谓格式化字符串，就是在*printf()系列函数中按照一定的格式对数据进行输出。*printf()系列函数有 3 条特殊的性质，这些特殊性质如果被攻击者结合起来利用，就会形成漏洞。利用这些漏洞，攻击者可以改变程序运行的正常流程，实现其恶意行为。假设一个打开调用失败，日志记录其文件名。攻击者利用这种漏洞，当程序试图打开文件但打开失败时，攻击者使用代码将自己伪装成打开调用的一个参数，其中包含了一个长度有几百个字节的字符串，用文件名触发格式化字符串攻击。

为了确定正常的字符串长度，在“学习”阶段，对字符串参数的长度样本 $l_1, l_2, \cdots, l_n$ 计算样本均值 $\overline{X}$ 、样本方差 σ^2 来近似真实字符串的均值 $\hat{X}$ ，方差 $\hat{\sigma}^2$ 。在“检测”阶段，实际系统调用的参数值与已建立的 Profile 比较，确定检测的这些参数值是否在合理的范围内。

9.5.2　应用程序层的异常入侵检测方法

应用程序层的异常入侵检测系统是基于正常的应用程序行为创建一个应用程序行为的档案（Profile）。这个行为表现为应用程序执行的操作。Profile 可以通过观测正常行为的踪迹来创建，也可以由一个应用程序允许执行的所有确定操作来建立。

例 9.12　跟踪数据库事务进行数据库入侵检测的系统 DIDAFIT（Detecting Intrusions in Databases Through Fingerprinting Transactions）。该系统首先建立一个应用程序合法访问数据库的指纹集合，通过一个程序收集 SQL statements，而指纹是对每个 SQL

statement 的特征描述，并用来检测异常的 statements。该系统通过跟踪数据库合法事务访问模式来识别数据库入侵行为 （特别是 SQL 注入攻击）。该系统解决了从数据库指纹日志中学习合法指纹集合的问题，这些数据库指纹日志中包含了合理事务的 SQL statements。有人改进了该系统，将 SQL statements 总结为更全面的指纹，可以得到训练数据中缺少的但可能是合理的指纹，并且能够识别在训练集合中可能是恶意的 SQL statements。

例 9.13　基于 Web 应用程序分析 Web 客户端交互行为的入侵检测系统。

该系统参照服务器端的程序分析客户端的查询，针对这些查询的不同特征建立模型。例如，服务器端程序访问模式触发访问的各个参数值。结合 Web 应用程序自动推得参数的档案（Profile）（如请求参数的长度和结构）。另外，监测应用程序查询和访问模型在时间上的关系。访问模式的变化预示着攻击行为。当一个通常情况下很少被访问的应用程序突然暴露出大量的调用，这种情况可能是攻击者插入攻击探针或是猜测参数值的试探。

另一种模式关注程序访问的顺序。Web 应用程序通常由一组服务器端程序组成，这些程序共同完成应用程序功能。例如，一个商店应用程序有一个登录程序授权一个用户，一个程序去访问一个目录，一个程序在虚拟的购物车中加入一些项目，一个程序执行校验和付款。Web 应用程序的特点是可以强制它的程序按照明确定义的顺序执行。例如一个用户首先要登录成功才能执行其他的事务。不正常的程序访问顺序可能预示着恶意行为，如攻击者企图绕过登录检查直接去访问一个特权程序。

9.6　文件完整性检查

文件完整性检查（Integrity Checkers）是一种重要的基于主机的入侵检测工具，这个工具的任务是检测某个观察的文件是否被篡改。通常攻击者在入侵时都会对一些文件进行改动，因此通过检测文件内容的非法更改可以判定入侵，它经常用来识别攻击者成功入侵后的行为。一个著名的完整性检查工具是 TripWire。

例 9.14　利用文件完整性检查监测 Rootkit 攻击。

首先来了解什么是 Rootkit？攻击者成功入侵之后通常会在入侵主机上安装一个 Rootkit。Rootkit 是攻击者用来隐藏自己的踪迹和保留“root”访问权限的工具。一旦拥有了“root”级的权力，入侵者就能修改系统指令去隐藏它的痕迹，以防系统管理员发现，并保留其根目录访问权。还可以收集系统和环境的信息，确保今后的访问（如服务器后门）。Rootkit 修改系统审计程序，不向管理员返回任何攻击者使用过的特定文件和进程的信息。文件完整性检查可以监测到 Rootkit 替换或修改系统文件及入侵者人工修改的文件。因此，利用文件完整性检查，可以有效地检测到 Rootkit 攻击。

文件系统完整性检查的缺陷是只能在离线状态可靠地执行分析。也就是说，文件系统需要被已知没有受到侵害的操作系统引导。原因是完整性检查依赖于操作系统去访问文

件系统，如果操作系统自身在入侵主机上遭到破坏，则完整性检查可能提供错误信息。

文件完整性检测技术属于一种事后的入侵检测技术，并不具备对入侵行为检测的实时性，它检测一遍文件系统的时间取决于文件系统的空间大小。尽管基于文件完整性检验的检测技术存在着一些缺点，但是大多数入侵行为都会对文件系统进行非法修改，这时它就发挥作用了，尤其是当滥用检测和异常检测发生漏报情况时，文件完整性检验的检测结果为入侵检测提供重要依据。

9.7　本章小结

本章首先介绍了入侵检测的概念、系统模型、作用和分类，使读者对入侵检测技术有了初步的认识。接着重点介绍基于主机的入侵检测各个方面的情况，包括基于主机入侵检测技术的发展历程和发展趋势；基于主机入侵检测的信息源；基于主机入侵检测的方法。通过本章的学习，使读者对基于主机的入侵检测有一个深入的了解。

基于主机入侵检测的信息源主要来自操作系统的审计记录、系统日志和其他应用程序的日志。对于不同类型的操作系统，其审计记录的格式和获取方法是不同的，本章以 UNIX、Linux 和 Windows 操作系统为例，介绍了不同操作系统的审计记录。对于系统日志，本章以 UNIX 操作系统为例进行了介绍。对于不同应用程序其日志信息也是不一样的，以 FTP 和 Web 为例具体介绍了应用程序的日志信息在入侵检测中的作用。

对于基于主机的入侵检测方法，本文主要从滥用入侵检测方法和异常入侵检测方法来进行讨论。对于滥用入侵检测方法，在操作系统层采用了基于状态转移图，利用 STATL 描述攻击场景的方法；在应用程序层利用应用程序的日志文件来检测入侵行为。对于异常入侵检测方法，在操作系统层，以系统调用为审计数据进行程序行为的异常检测；在应用程序层，基于正常的应用程序行为创建一个应用程序行为的档案，任何与应用程序正常行为档案相违背的操作都视为入侵。

另外，利用文件完整性检测对那些更改文件的入侵行为进行检测，也可以较好地实现基于主机的入侵检测。



习题 9

一、选择题

1．入侵检测系统的两个主要类型是（　　）。

A．基于特征的和基于事件的　　B．基于网络的和基于主机的

C．主动的和被动的　　D．智能的和被动的

2．基于主机的入侵检测（　　）。

A．防火墙活动记录，　　B．蜜罐内的活动记录

C．本地主机上的活动　　D．网络探测设备截取的信息

3．蜜罐主要用于（　　）。

A．收集法律证据　　B．用虚拟目标来研究攻击者的行为

C．处理防火墙和路由器的事件　　D．吸引顾客来到电子商务网站

二、简答题

1．什么是入侵检测？

2．入侵检测的作用体现在哪些方面？

3．基于主机的入侵检测系统和基于网络的入侵检测系统的区别是什么？各有什么优缺点？

4．滥用入侵检测系统的设计原理是什么？

5．异常入侵检测系统的优缺点是什么？

6．基于主机入侵检测系统的数据源主要有哪些？

7．简述 STAT 模型的原理。

8．举例说明基于主机的异常入侵检测方法。

第 10 章
计算机病毒原理及其防治

本章将介绍计算机病毒的基本知识，包括计算机病毒的定义、特征、分类、工作机制、传播和危害。然后给出目前的一些计算机病毒的防治技术，包括计算机病毒的清除、检测和预防。

10.1　计算机病毒的概念

计算机病毒正在成为一个越来越普遍的问题，每个使用过计算机的人都遇到过某种形式的计算机病毒。先来看几个著名的计算机病毒的例子。

（1）CIH 病毒：1998 年 8 月份从中国台北传入中国大陆的 CIH 病毒共有三个版本：1.2 版、1.3 版、1.4 版。它们的发作时间分别是 4 月 26 日、6 月 26 日和每月 26 日。该病毒是第一个直接攻击、破坏硬件的计算机病毒，也是破坏最为严重的病毒之一。它主要感染 Windows 95/98 的可执行程序，破坏计算机 Flash BIOS 芯片中的系统程序，导致主板损坏，同时破坏硬盘中的数据。当病毒发作时，硬盘驱动器不停旋转，硬盘上所有数据（包括分区表）被破坏，只有对硬盘重新分区才有可能挽救硬盘。同时，病毒能够直接攻击某些主板，会将 Flash BIOS 中的系统程序破坏，造成开机后系统无反应。1999 年 4 月 26 日，CIH 病毒在全球范围大规模爆发，造成近 6000 万台计算机瘫痪。中国也未能在这次灾难中幸免，直接经济损失达 8000 万元，间接经济损失超过了 10 亿元。该病毒给整个世界带来的经济损失在数十亿美元以上。

（2）Happy99 病毒：1999 年出现的 Happy99 病毒是完全通过 Internet 传播的病毒，标志着 Internet 病毒将成为病毒新的增长点。第一次执行 Happy99 病毒时，它会表现为一幅放礼花的图像。通常该程序会以邮件的附件形式传递开来，或者从一些程序组中下载而来。机器感染病毒后，病毒自我复制，并加载病毒文件发送给自己的好友。Happy99 病毒的特点就是利用 Internet 的优势，快速进行大规模的传播，从而使病毒在极短的时间内遍布全球。

（3）红色代码（Code Red）病毒：2001 年 7 月中旬，一种名为“红色代码”的病毒在美国大面积蔓延，这个专门攻击服务器的病毒攻击了白宫网站，造成了全世界的恐慌。8 月初，其变种“红色代码Ⅱ”针对中文系统作了修改，增强了对中文网站的攻击能力，开始在中国蔓延。“红色代码”病毒通过一种黑客攻击手段，利用服务器软件的漏洞来传播，它造成了全球 100 万个以上的系统被攻陷从而导致瘫痪。这是计算机病毒与网络黑客首次结合，对后来的病毒产生了很大的影响。

（4）冲击波病毒（Worm Blaster）：计算机感染该病毒后，会导致系统运行不稳定。RPC 漏洞导致 Windows 2000、Windows XP 的机器非常不稳定，会出现计算机重启、死机等异常现象。2003 年 8 月，Blaster 爆发，数十万台计算机被感染，给全球造成 20 亿～100 亿美元的损失。

从以上的例子可以看出计算机病毒已经成为现代信息社会中最不安全的因素之一。

那什么是计算机病毒呢？在生物学中，病毒是指侵入动植物体等有机生命体中的具有感染性、潜伏性、破坏性的微生物，而且不同的病毒具有不同的诱发因素。人们联系到破坏计算机系统的“病原体”具有与生物病毒相似的特征，借用生物学病毒而使用了“计算机病毒”这个计算机术语。

早在计算机刚刚诞生的时候，就有了计算机病毒的概念。1949 年，计算机之父冯 • 诺依曼在《复杂自动机组织论》中便定义了计算机病毒的概念——一种“能够实现复制自身的自动机”。

计算机病毒这个词语最早是出现在科幻小说里的。1977 年夏天，托马斯 • 简 • 瑞恩（Thomas.J.Ryan）的科幻小说《P-1 的春天》（The Adolescence of P-1）成为美国的畅销书。作者在这本书中描写了一种可以在计算机中传染的病毒，病毒最后控制了 7000 台计算机，造成了一场灾难。不过，这在当时并没有引起人们的注意。1983 年，美国计算机安全专家 Frederick Cohen 博士首次提出计算机病毒的存在。1989 年，他进一步将计算机病毒定义为“病毒程序通过修改其他程序的方法将自己的精确复件或可能演化的形式放入其他程序中，从而感染它们”。所谓感染是指计算机病毒将自身嵌入到指令序列中，致使执行合法程序的操作导致病毒程序的共同执行。1994 年，在《中华人民共和国计算机安全保护条例》中明确定义：“计算机病毒是指编制或在计算机程序中插入的、破坏计算机功能或数据、影响计算机使用并能自我复制的一组计算机指令或程序代码”。

从广义上讲，计算机病毒是一种人为制造的、能够进行自我复制的、对计算机资源具有破坏作用的一组程序或指令的集合。近年来，将特洛伊木马、网络蠕虫等恶意代码均看做病毒，这与众多杀毒软件对病毒的定义是一致的。但如果要严格区分，这些恶意代码并不感染其他正常程序，而是通过持续不断地反复复制自己、增加自己的复件数量、消耗系统资源（如内存、磁盘存储空间、网络资源等），最终导致系统崩溃。本章会将这些恶意代码的概念在单独的章节里进行讨论。

10.2　计算机病毒的发展历程及发展趋势

10.2.1　计算机病毒的发展历程

通常认为计算机病毒起源于一种电子游戏——“磁芯大战”（Core War）。20 世纪 60 年代初，美国贝尔实验室的三个年轻程序员受到冯 • 诺依曼理论的启发，发明了“磁芯大战”游戏。游戏玩法如下：双方各写一套程序并将程序输入到同一台计算机中，这两套程序在计算机系统内互相追杀，有时它们会放下一些关卡甚至会停下来修复被对方破坏的几行指令。当它被困时，也可以把自己复制一次从而逃离险境，因为它们都在计算机的记忆磁芯中游走，所以得到了“磁芯大战”之名。双方的程序在指令的控制下会竭力去消灭对方的程序，在预定的时间内，谁的程序繁殖得多，谁就获胜。这种有趣的游

戏很快就传播到其他计算机中心。

1983 年，计算机病毒首次在计算机界被确认存在。1983 年 11 月 3 日，Frederick Cohen 博士研制出一种在运行过程中可以复制自身的破坏性程序，Len Adleman 将它命名为计算机病毒（Computer Viruses），并在每周一次的计算机安全讨论会上正式提出。同一天，专家们在 VAX11/750 计算机系统上验证了计算机病毒的存在。

1986 年初，在巴基斯坦的的两兄弟 Basit 和 Amjad 编写了巴基斯坦（Pakistan）病毒，即 Brain 病毒。Brain 病毒首次使用了伪装的手段来迷惑计算机用户。1987 年美国 Delawre 大学的计算机首先遭受到该病毒的攻击，之后 Brain 病毒席卷全球。

1987 年，计算机病毒主要是引导型病毒，首次出现了能自我加解密的病毒——Cascade，另外有代表性的是 PingPong 和 Stoned 病毒。同年 12 月，出现第一个网络病毒——Christmas Tree。

1988 年 6 月 13 日，这天正好是星期五，一些国家的公司和大学遭到了“耶路撒冷（Jerusalem）”病毒的攻击。该病毒因攻击了耶路撒冷大学而得名，又因为它是既是 13 号又是在星期五这天爆发的，又被称为“黑色星期五”。同年的 11 月 2 日，发生了“Morris Worm”病毒事件。美国康奈尔大学的学生——23 岁的罗伯特·莫里斯（Robert.T.Morris）将自己编制的蠕虫程序输入到计算机网络中，在几个小时内造成互联网堵塞，6000 多台计算机被感染，造成巨大的损失。迫使美国政府迅速做出反应，成立了 CERT（计算机应急行动小组）。这是计算机问世以来最为严重的计算机病毒攻击事件，其制造者受到了美国法律的制裁。同时，反病毒技术也已经开始成熟，Doctors Solomon’s Anti-Virus Toolkit 成为当时最强大的反病毒软件。

1989 年年初，我国某市统计局计算机中心的 M24 微型计算机受到 Brain 病毒的攻击，这是国内首例病毒攻击事件。7 月中国公安部推出了中国最早的杀毒软件 Kill6.0，这是国内推出的第一个杀毒软件。这一年，可执行文件型病毒出现，在系统执行文件时病毒取得控制权，并将自己附加在可执行文件中，使文件长度增加。例如，Eddie 病毒可以驻留内存，不仅感染被调用的文件，还感染被打开或被复制的文件；Frodo 病毒是第一个全秘密寄生的文件病毒，它不改变被感染文件的长度；同时出现了名为 AIDS 的特洛伊木马型病毒。

1990 年 1 月，Mark Washburn 研制出了第一个多态病毒——Chameleon，该病毒也称 V2P1。同年保加利亚的程序员开发出了第一个“病毒自动生成加工工厂”软件——“Virus Production Factory”。建立了第一个为病毒制造者服务的 BBS。同年，中国深圳华星公司推出了基于硬件的反病毒系统——“华星防病毒卡”。

1991 年 1 月 8 日，美国国家安全局研制的“AF/91”病毒使伊拉克防空系统 50%的打印机和显示器瘫痪（时逢 1990—1991 年海湾战争期间）。这一年发现的比较有名的病毒有能够感染文件和引导区的复合多态病毒 Tequila。同时反病毒公司也得到了发展壮大，Symantec 公司（其著名的反病毒工具软件为诺顿 Norton）和 Central Point 公司（其

著名的软件为 Pctools）各自推出了自己的查杀毒软件。中国瑞星公司宣告成立，瑞星防病毒卡面市。

1992 年，第一个多态病毒生成器——MTE 开发成功，第一个病毒构造工具集——Virus Construction Sets 和病毒创建库——Virus Create Library 开发成功。促使更多的病毒被源源不断地制造出来。出现了针对杀毒软件的“幽灵”病毒，如 One-half。这一年，在芬兰发现了首例 Windows 病毒。

1994 年，出现了第一个感染对象模块文件（即.obj 文件）的 Shifter 病毒。出现了第一个感染 C 语言和 PASCAL 语言源代码的 Srcvir 源码病毒。出现了一种新的引导型病毒——3APA3A，该病毒能够对抗当时所有的查杀毒软件。

1995 年 1 月，微软第一次发现在其发行的光盘中含有病毒。同年 8 月 9 日，在美国首次发现专门攻击 Word 文件的新病毒——宏病毒 Concept。从此宣告了攻击 Windows 操作系统病毒的大规模出现。

1996 年，随着微软新的操作系统 Windows 95、Windows NT 和微软办公软件 Office 的流行，病毒制造者开始面向新的环境，制造出了许多新的病毒。1 月出现了第一个 Windows 95 病毒——Win95.Boza。7 月出现了第一个 Microsoft 电子数据表病毒——Laroux。

1997 年 2 月出现第一个 Linux 环境下的 Bliss 病毒，结束了 Linux 系统从未被病毒感染的历史。同年 4 月第一个利用文件传输协议 FTP 进行传播的蠕虫病毒——Homer 出现。这一年有大量的宏病毒流行，被公认为计算机反病毒界的“宏病毒”年。宏病毒主要感染 Word、Excel 等文件。如 Word 宏病毒，早期是用一种专门的 Basic 语言即 WordBasic 所编写的程序，后来使用 Visual Basic。与其他计算机病毒一样，它能对用户系统中的可执行文件和数据文本类文件造成破坏。常见的宏病毒有台湾一号（Tw No.1）、Setmd、Consept、Mdma 等。

1998 年 6 月，被称为有史以来影响最大、危害最大的病毒——CIH 病毒出现，该病毒是第一个能够直接攻击某些主板上的 Flash ROM 中 BIOS 信息的文件型病毒，它既攻击硬盘中的文件系统，又攻击计算机主板，并使其破坏。同年 8 月，出现了第一个攻击 Java 可执行文件的 Strange Brew 病毒。

1999 年，通过邮件进行病毒传播开始成为病毒传播的主要途径，出现了第一个在互联网上大规模传播的网络蠕虫病毒 Happy99。出现了一种宏病毒和蠕虫的混合物 Melissa，通过电子邮件系统大量传播，造成网络的阻塞、瘫痪。4 月 4 日，Melissa 病毒及其变种攻击北约的通信系统，美国海军陆战队所有作战单元的电子邮件服务器被阻塞。

2000 年，被称为 VBScript 病毒/蠕虫之年。大量使用脚本技术的病毒出现，5 月，出现了利用 Microsoft Outlook 进行传播的脚本病毒——LoverLetter，它给电子邮件的附件两个扩展名（Love-Letter-for you .txt .vbs），得到了比 Melissa 病毒更快更远的传播。

2001 年 7 月，一种新型的网络病毒——Code Red 席卷美国，8 月出现了 Code Red

的变种“Code Red II”，9 月出现了 Nimda 病毒。造成了大范围的互联网上服务器被阻塞或访问速度下降，在世界范围内造成了巨大的损失。仅 Code Red 病毒所造成的经济损失就远远超过过去六年来任何一年的年度损失。Code Red 利用 IIS 漏洞，开创了黑客与病毒并肩作战的攻击模式，使传统的防病毒软件面临更大的挑战。

2002 年，病毒主要利用网络及电子邮件进行传播。其中求职信（Klez）病毒是杰出代表，不仅有英文版和中文版两种形式，并且有多种变种。它能自动获取用户地址薄中的信息乱发邮件，甚至每次发送病毒邮件的主题和内容都是随机的，危害程度极大。

2003 年，“2003 蠕虫王”病毒在亚洲、美洲、澳大利亚等地迅速传播，造成了全球性的网络灾害。其中受害最严重的无疑是美国和韩国这两个互联网发达的国家。其中，韩国 70%的网络服务器处于瘫痪状态，网络连接的成功率低于 10%，整个网络速度极慢。美国不仅公众网络受到了破坏性的攻击，而且连银行网络系统也遭到了破坏，全国 1.3 万台自动取款机处于瘫痪状态。8 月，冲击波（Blaster）病毒爆发。它是一种网络蠕虫，主要感染的操作系统是 Windows 2000 和 Windows XP 系统。冲击波病毒是继 Code Red 病毒以来影响最大的漏洞攻击型病毒，数十万台计算机被感染，给全球造成 20 亿～100 亿美元的损失。

2004 年 1 月 26 日，“我的末日/悲惨命运（MyDoom）”病毒在互联网上出现，并迅速大规模传播。该病毒阻止被传染的计算机访问微软的某些网站，并对微软的主要网站实施 DoS 攻击。1 月 29 日，微软宣布悬赏 25 万美元缉拿 MyDoom 病毒的个人或团体。五一期间及其后的短短十几天内，“震荡波（Sasser）”病毒席卷了全世界，数千万计算机瘫痪，数亿财产在这次浩劫中遭到损失。该病毒由 18 岁的德国少年斯文•扬森编写，它与 2003 年的“冲击波”病毒类似，也属于网络蠕虫，通过互联网进行传播，但不通过邮件传播。

2005 年是木马流行的一年。经典木马病毒如赛波（SdBot.atp.Rootkit）、BO2K、冰河、灰鸽子（Huigezi）等。它们都可以盗取用户信息，主要以获取经济利益为目的。它们在后台运行，没有任何提示，偷偷记录用户的输入信息，如 QQ 密码、网络游戏账号、网上银行卡账号等，并将这些信息直接发送到黑客手中，给用户带来直接经济损失。2005 年截获的“盗号木马”的数量是 5484 个。

2006 年木马仍然是病毒主流，其变种层出不穷。灰鸽子等病毒仍旧肆虐，造成了不小的损失。

2007 年 1 月，肆虐的“尼姆亚（Worm Nimaya）”（又名“熊猫烧香”）病毒成为当年的“毒王”。这一病毒通过感染文件传播，可造成文件损坏。该病毒是一个能在 Win9X/NT/2000/XP/2003 系统上运行的蠕虫病毒。这一病毒采用“熊猫烧香”头像作为图标，诱使计算机用户运行。它的变种会感染计算机上的 .exe 可执行文件，被病毒感染的文件图标均变为“熊猫烧香”。同时，受感染的计算机还会出现蓝屏、频繁重启及系统硬盘中数据文件被破坏等现象。该病毒会在中毒计算机中所有的网页文件尾部添加病

毒代码。一些网站编辑人员的计算机如果被该病毒感染，上传网页到网站后，就会导致用户浏览这些网站时也被病毒感染。

据江民反病毒预警中心、江民全球病毒监测网提供的数据显示，截至 2007 年 12 月，江民反病毒中心共截获计算机新病毒总数为 363 000 余种，这一数字是 2006 年的 6 倍多。2007 年全年病毒累计感染计算机 34 414 793 台，新病毒感染计算机台数为 28 879 529 台，其中木马病毒感染 24 875 117 台，占病毒总数的 72.28%；后门病毒感染 1 987 821 台，占病毒总数的 5.78%；蠕虫病毒感染 1 556 124 台，占病毒总数的 4.52%；流氓软件感染 2 502 563 台，占病毒总数的 7.27%；脚本病毒感染 80 026 台，占病毒总数的 0.23%。

10.2.2　计算机病毒的发展阶段

IT 行业普遍认为，从最原始的单机磁盘病毒到现在逐步进入人们视野的手机病毒，计算机病毒主要经历了六个重要的发展阶段。

第一阶段为原始病毒阶段。产生年限一般认为在 1986—1989 年之间，由于当时计算机的应用软件少，而且大多是单机运行，因此病毒没有大量流行，种类也有限，病毒的清除工作相对来说较容易。主要特点是：攻击目标较单一；主要通过截获系统中断向量的方式监视系统的运行状态，并在一定的条件下对目标进行传染；病毒程序不具有自我保护的措施，容易被人们分析和解剖。

第二阶段为混合型病毒阶段。其产生的年限在 1989－1991 年之间，是计算机病毒由简单发展到复杂的阶段。计算机局域网的应用与普及给计算机病毒带来了第一次流行高峰。这一阶段病毒的主要特点为：攻击目标趋于混合；采取更隐蔽的方法驻留内存和传染目标；病毒传染目标后没有明显的特征；病毒程序往往采取了自我保护措施；出现许多病毒的变种，等等。

第三阶段为多态性病毒阶段。此类病毒的主要特点是：在每次传染目标时，放入宿主程序中的病毒程序大部分都是可变的。因此防病毒软件查杀时非常困难。如 1994 年在国内出现的“幽灵”病毒就属于这种类型。这一阶段病毒技术开始向多维化方向发展。

第四阶段为网络病毒阶段。从 20 世纪 90 年代中后期开始，随着国际互联网的发展壮大，依赖互联网传播的邮件病毒和宏病毒等大量涌现，病毒传播快、隐蔽性强、破坏性大。也是从这一阶段开始，反病毒产业开始萌芽并逐步形成一个规模宏大的新兴产业。

第五阶段为主动攻击型病毒。典型代表为 2003 年出现的“冲击波”病毒和 2004 年流行的“震荡波”病毒。这些病毒利用操作系统的漏洞进行进攻型的扩散，并不需要任何媒介或操作，用户只要接入互联网就有可能被感染。正因为如此，该病毒的危害性更大。

第六阶段为“手机病毒”阶段。随着移动通信网络的发展及移动终端——手机功能的不断增强，计算机病毒开始从传统的互联网走进移动通信网络世界。与互联网用户相

比，手机用户覆盖面更广、数量更多，因而高性能的手机病毒一旦爆发，其危害和影响比“冲击波”“震荡波”等互联网病毒还要大。

10.2.3　计算机病毒的发展趋势

从病毒的整个发展过程可以看出，计算机及其相关技术的发展是病毒技术发展的基础。根据近期病毒的特征可以看出新世纪计算机病毒具有以下新的发展趋势：网络化、专业化、智能化、人性化、隐蔽化、多样化和自动化。

1．网络化

新时期的病毒充分利用计算机技术和网络技术的交叉点。自 2002 年以来，通过网络漏洞和邮件系统进行传播的蠕虫病毒就开始成为新宠，数量上已经远远超过了曾经是主流的系统病毒，并且得到了更快的发展，成为不折不扣的主流病毒。在 2003 年和 2004 年的计算机流行病毒列表中，有一半以上是蠕虫病毒。2005 年至今，木马成为了最流行的恶意代码。这些病毒的迅速发展说明了目前国内网络建设速度加快，但网络安全却未及时跟上，网络防毒将成为今后网络管理工作的重点。电子邮件无疑是目前互联网时代最主要的信息沟通方式，这种方式导致了邮件病毒迅速增长，一些蠕虫、木马、恶意程序等纷纷利用该平台进行传播。

2．专业化

手机病毒、PDA 病毒的出现标志着病毒开始向专业化方向发展。由于这些设备都采用嵌入式操作系统且软件接口较少，所以以往很少有病毒制造者涉足这个领域。随着时间的推移和技术细节的公开，已经有人开始转向这个领域。

3．智能化

与传统的计算机病毒不同的是，许多新病毒、蠕虫、黑客工具、木马等恶意程序是利用当前最新的编程语言与编程技术实现的，并且易于修改以产生新的变种，从而逃避反病毒软件的搜索。例如，“爱虫”病毒是用 VBScript 语言编写的，只要通过 Windows 下自带的编辑软件修改病毒代码中的一部分，就能轻而易举地制造病毒变种，以躲避反病毒软件的追击。另外，新病毒可以利用 Java、ActiveX 及 VBScript 等技术，潜伏在 HTML 页面里，在上网浏览时触发。Kakworm 病毒虽然早在 2006 年 1 月就被发现，但它的感染率一直居高不下，原因是由于它利用 ActiveX 控件中存在的缺陷传播，装有 IE 5 或 Office 2000 的计算机都可能被感染。这个病毒的出现使原来不打开带毒邮件附件而直接删除的防邮件病毒方法完全失效。更令人担心的是，一旦这种病毒被赋予其他计算机病毒的恶毒的特性，所造成的危害很有可能超过任何现有的计算机病毒。

4．人性化

人性化，或者说是诱惑性更确切一些。现在的计算机病毒越来越注重分析人类的心

理，如好奇、贪婪等。曾肆虐一时的“裸妻”病毒，主题就是英文的 Naked Wife，邮件正文为“我的妻子从未这样”，邮件附件中携带一个名为“裸妻”的可执行文件，用户执行这个文件，病毒就被激活。最近出现的 My-babypic 病毒，通过可爱宝宝的照片传播病毒。而“库尔尼科娃”病毒的大流行，则是基于网坛美女库尔尼科娃挡不住的魅力。

5．隐蔽化

相比较而言，新一代病毒更善于隐藏自己、伪装自己。病毒主题会在传播中改变，或者有极具诱惑性的主题、附件名；许多病毒会伪装成常用程序，或者将病毒代码写入文件内部并让文件长度不发生变化，使用户防不胜防。例如，主页病毒的附件 homepage.html.vbs 并非一个 HTML 文档，而是一个恶意的 VB 脚本程序，一旦执行，就会向用户地址簿中的所有电子邮件地址发送带毒的电子邮件副本。再如维罗纳病毒，将病毒写入邮件正文，而且主题、附件名极具诱惑性，主题众多，更替频繁，使用户因麻痹大意而感染。而 matrix 等病毒会自动隐藏、变形，甚至阻止受害用户访问反病毒网站和向病毒记录的反病毒地址发送电子邮件，无法下载经过更新、升级后的相应杀毒软件或发布病毒警告消息。

6．多样化

新病毒层出不穷，老病毒也充满活力，并呈现多样化的趋势。1999 年普遍发作的计算机病毒分析显示，虽然新病毒不断产生，但较早的病毒发作仍很普遍。1999 年报道最多的病毒是 1996 年就首次发现并到处传播的宏病毒 Laroux。新病毒可以是可执行程序、脚本文件、HTML 网页等多种形式，并正向电子邮件、网上贺卡、卡通图片、ICQ、OICQ 等方面发展。更为棘手的是，新病毒的破坏性更强。据计算机经济研究中心的报告显示，在 2000 年 5 月，“爱虫”病毒大流行的前 5 天，就造成了 67 亿美元的损失。而该中心 1999 年的统计数据显示，到 1999 年年末，病毒损失才达 120 亿美元。

7．自动化

以前的病毒制作者都是专家，编写病毒在于表现自己高超的技术。但是“库尔尼科娃”病毒的设计者不同，他只是修改了下载的 VBS 蠕虫孵化器，“库尔尼科娃”病毒就诞生了。据报道，VBS 蠕虫孵化器被人们从 VXHeavens 上下载了 15 万次以上。正是由于这类工具太容易得到，使得现在新病毒出现的频率超出以往任何时候。随着病毒技术的发展，计算机病毒的数量也在迅速增加。VirusScan 声称现在大约有 57 000 多种不同形式的计算机病毒存在。很多公司都认为病毒数量已经远远超过 50 000 种。

10.3　计算机病毒的表现形式

计算机病毒是一段特殊的程序，它与正常程序的区别如下。

- 正常程序是具有应用功能的完整程序，以文件形式存在，具有合法的文件名；

而病毒一般不以文件形式独立存在，一般没有文件名，它隐藏在正常程序和数据文件中，是一种非完整的程序。

- 正常程序依照用户的命令执行，完全在用户的意愿下完成某种操作，也不会自身复制；而病毒在用户完全不知道的情况下运行，将自己复制到其他正常程序中，与合法程序争夺系统的控制权，进行各种破坏活动。
- 正常的计算机程序一般是不会将自身的代码强行链接到其他程序之上的，而病毒却能使自身的代码强行传染给一切符合其传染条件的程序之上。

在一般情况下，计算机病毒总是依附某一系统软件或用户程序进行繁殖和扩散，病毒发作时危及计算机的正常工作、破坏数据与程序、侵犯计算机资源。计算机在感染病毒后，总是有一定规律地出现一些异常现象。

（1）屏幕显示异常。屏幕显示出的不是由正常程序产生的画面或字符串，屏幕显示混乱。例如，众所周知的小球（Ball）病毒，在被感染的计算机系统的显示屏上显示一个像乒乓球一样的白点，跳来跳去，并在屏幕的四周边界弹性碰撞。“女鬼”病毒发作时显示骷髅动画。

（2）系统发出异常响声。计算机的内置喇叭一般不会发出声音，除非运行了特殊的程序。有些计算机病毒发作时会发出奇怪的声音。例如，当感染 MS Word97 文档的 W97m.zhao 病毒发作时，不断有蜂鸣声发出。水泥匠（Plasterers）病毒发作时，计算机的喇叭会发出断断续续的炸弹爆炸的声音。

（3）系统引导时间增加、运行速度明显下降。例如，W97m.twno.a 病毒在每月的 13 日向用户提出一些心算题目，若用户解答错误，系统则会给每道题打开 20 个文档，使 Windows 的运行速度变慢。

（4）用户没有访问的设备出现工作信号。例如，V.8888 病毒能使被该病毒感染的硬盘指示灯一直亮着。

（5）磁盘出现莫名其妙的文件和坏块。占用磁盘上的可用空间，使磁盘可用空间减少。例如，W97m.class.d 病毒在每年的 12 月 25 日被激活时，将毁坏 CMOS 数据，向染毒的机器的所有文件写入垃圾数据，并破坏 Flash BIOS。

（6）系统不能识别硬盘或硬盘不能引导系统。如果病毒感染了硬盘的主分区表，或者导致分区表丢失，都会造成开机不承认硬盘或系统引导失败。

（7）不应驻留内存的程序驻留内存，内存空间减小，系统运行速度下降。病毒发作后常驻内存，并且迅速自我复制，导致内存空间不足，并且占用 CPU 资源，使系统运行速度明显下降。例如，One-Half 病毒就是一个多态的、常驻内存的加密病毒。对于 W97m.class.d 病毒，当每次启动 Windows 后，该病毒立即变为常驻内存的。

（8）系统经常无端死机或自动重启。

（9）磁盘访问时间比平时增加，或者对整张磁盘或盘上某个扇区进行格式化，破坏磁盘数据。覆盖磁盘的启动磁道和目录表，使磁盘变成废盘。

（10）修改磁盘卷标。Brain 病毒将系统磁盘的卷标修改为 Brain。

（11）鼠标失控。屏幕上的光标自己移动，应用程序自己运行，有被遥控的现象。

（12）可执行文件被无端删除。例如，W97m.mjulyKiller.d 病毒会迫使用户在病毒发作时在弹出的“醒世恒言”对话框中单击“确定”按钮，表示与其观点一致，如果用户 3 次单击“取消”按钮，则病毒会将 C 盘根目录下的 Autoexec.bat 自动批处理文件的内容全部删除，再写入一个破坏性极强的命令代码。当用户再次启动计算机时，C 盘根目录下所有文件被删除，导致计算机瘫痪。

（13）修改、删除某些系统文件，系统配置出现错误。例如，W97m.newhope.a 病毒往系统文件 Autoexec.bat 中增加代码。Troj.Bo2k 病毒能将原来的 Autoexec.bat 复制成为 Ae.Kak，然后修改 Autoexec.bat 并覆盖.Kak 与 .hta 文件。

（14）文件的长度、建立日期或属性发生变化。例如，W97m.Laroux.a 病毒能删除被它感染的文件的属性内容（如标题、主题、作者、关键字和内容等）。幽灵（Ghost）病毒使每个感染文件长度增加 3544B。

（15）文件数目变化。例如，Macho soft 病毒不驻留内存，但自身加密，感染.com 和.exe 文件，病毒发作后，被感染磁盘的根目录下会增加一个名为 ibmnetio.sys 的隐藏文件。

（16）发出虚假报警。例如，Best Wishes 病毒主要感染.com 文件，每当查看或运行病毒文件时，都会发出虚假报警信息：“File not found!”。

（17）异常要求用户输入口令（password）。例如，W97m.malaysia 病毒在每年的 8 月 30 号发作时，会要求用户输入一个口令（正确的口令为 WM.MALAYSIA1998）。

（18）键盘工作异常。例如，含有 23 个宏的 W97m.lulung.b 病毒将会使键盘的 ALT+F8 组合键、ALT+F11 组合键失效。Attention 病毒别名为 User394 病毒，该病毒驻留内存后，计算机用户每次单击键盘时系统的扬声器都会发出“咔哒”声。

（19）打印机工作异常。打印机的工作速度减慢，有时产生死锁现象。例如，Typo Boot 病毒又名 Mistake 病毒，此病毒发作时，使打印输出中的某些字符总是被其他语言相似的字符代替，数字也可能被另外的数字替换。1024 SBC 病毒又名 Ontario II 病毒、SBC 病毒，该病毒使被感染的文件长度增加，打印时出现间断性打印现象，系统可用内存减少。

上述的这 19 种表现形式并非包罗了现今已发现的大约 8 万种计算机病毒的所有情况，只能说是其中的一部分。从中能看出计算机病毒表现形式具有多样性和诱惑性的特点。这里的“诱惑性”是指利用人性的弱点，诱发用户的好奇心。例如，2000 年 5 月出现的 VBS.Love Letter.A 脚本病毒，其邮件主题是大写无空格的“ILOVEYOU”，包含一个名为“Love-Letter-fo-you.txt.vbs”的附件，用户好奇，在微软的电子邮件中打开这个附件，以致引发该病毒，造成损失。

10.4　计算机病毒的特征

计算机病毒种类繁多、特征各异，但一般具有以下特性：自我复制能力、很强的感染性、一定的潜伏性、特定的触发性、很大的破坏性。正确而全面地认识计算机病毒的特征有助于反病毒技术的研究。

10.4.1　计算机病毒的一般特征

1．程序性（可执行性）

从计算机病毒的定义可知，计算机病毒是一段具有特定功能的、严谨精巧的计算机程序。程序性既是计算机病毒的基本特征，也是计算机病毒最基本的一种表现形式。程序是由人来编写的，既然人能编写出计算机病毒程序，当然也能开发出反病毒程序，因此程序性也决定了计算机病毒的可防治性、可清除性。计算机病毒是一段可执行程序，但它不是一个完整的程序，而是寄生在其他可执行程序上的一段程序，因此它享有一切程序所能得到的权力和特点（如病毒程序必须驻留内存、必须经过编译之后形成目标代码、执行目标代码才能起作用等）。计算机病毒只有当它在计算机内得以运行时，才具有传染性和破坏性等特性，也就是说，计算机 CPU 的控制权是关键问题。若计算机在正常程序控制下运行，而不运行带病毒的程序，则这台计算机总是可靠的。在这台计算机上可以查看病毒文件的名字、查看计算机病毒的代码、打印病毒的代码，甚至复制病毒程序，却都不会感染上病毒。相反，计算机病毒一经在计算机上运行，在同一台计算机内病毒程序与正常系统程序争夺系统控制权时往往会造成系统崩溃，导致计算机系统瘫痪。

2．传染性

计算机病毒的传染性是指病毒具有把自身复制到其他程序的能力。传染性又称自我复制、自我繁殖、感染或再生。传染性是计算机病毒的最本质的属性，是判断一段计算机程序是否是计算机病毒的首要依据，也决定了计算机病毒的可判断性。

计算机病毒会通过各种渠道从已被感染的计算机扩散到未被感染的计算机，在某些情况下造成被感染的计算机工作失常甚至瘫痪。病毒可以感染文件、磁盘、个人计算机、局域网、互联网。以前，软盘和光盘是计算机病毒的主要感染载体；现在，网络（主要包括电子邮件、BBS、WWW 浏览器、FTP 文件下载等）成了计算机病毒的最主要的感染载体；点对点通信系统和无线通信系统则是最新出现的病毒感染载体。计算机病毒一旦进入计算机并得以执行，就会搜寻符合其传染条件的其他程序或存储介质，确定目标后再将自身代码插入其中，达到自我繁殖的目的。而被感染的目标又成了新的传染源，当它被执行以后，便又去感染另一个可以被其传染的目标。计算机病毒的

这种将自身复制到感染目标中的“再生机制”使病毒能够在系统中迅速扩散。

3．非授权性

非授权性是指计算机病毒未经授权而执行。一般正常的程序是由用户调用，再由系统分配资源，完成用户交给的任务，其目的对用户是可见的、透明的。而病毒隐藏在正常程序中，其在系统中的运行流程一般是：做初始化工作→寻找传染目标→窃取系统控制权→完成传染破坏活动，其目的对用户是未知的，是未经用户允许的。病毒的执行对系统来说是未授权的。

4．隐蔽性

计算机病毒一般是具有很高编程技巧、短小精悍的程序。通常附在正常程序中或磁盘较隐蔽的地方，也有个别的以隐含文件形式出现，目的是不让用户发现它的存在。如果不经过代码分析，病毒程序与正常程序是不容易区别开来的。一般在没有防护措施的情况下，计算机病毒程序取得系统控制权后，可以在很短的时间里传染给大量程序。而且受到传染后，计算机系统通常仍能正常运行，使用户不会感到任何异常。试想，如果病毒在传染到计算机上之后，机器马上无法正常运行，那么它本身便无法继续进行传染了。正是由于隐蔽性，计算机病毒得以在用户没有察觉的情况下扩散到上百万台计算机中。大部分病毒的代码之所以设计得非常短小，也是为了隐藏。病毒一般只有几百或1KB，而 PC 对 DOS 文件的存取速度可达每秒几百 KB 以上，所以病毒转瞬之间便可将这短短的几百字节附着到正常程序之中，使人非常不易被察觉。

5．潜伏性

计算机病毒的潜伏性是指其具有依附于其他媒体而寄生的能力。一个编制精巧的计算机病毒程序进入系统之后一般不会马上发作。潜伏性越好，其在系统中的存在时间就会越长，病毒的传染范围就会越大。潜伏性的第一种表现是指：病毒程序不用专用检测程序是检查不出来的，因此病毒可以静静地躲在磁盘或光盘里待上几天，甚至几年，一旦时机成熟，得到运行机会，就繁殖、扩散，继续为害。潜伏性的第二种表现是指：计算机病毒的内部往往有一种触发机制，不满足触发条件时，病毒具有一定的潜伏期，可以在系统中没有表现症状，从而不影响系统的正常运行。只有满足触发条件时，才会激活病毒的表现（破坏）模块而出现中毒症状。

6．可触发性

因某个事件或数值的出现而诱使计算机病毒实施感染或进行攻击的特性称为可触发性。为了隐蔽自己，病毒必须潜伏，少做动作。如果完全不动作、一直潜伏，病毒既不能感染也不能进行破坏活动，便失去了杀伤力。病毒既要隐蔽又要维持杀伤力，就必须具有可触发性。病毒的触发机制是用来控制感染和破坏动作的频率的。病毒运行时，触发机制检查预定条件是否满足，如果满足，则启动感染或破坏动作，使病毒实施感染或

攻击；如果不满足，则使病毒继续潜伏。病毒具有的预定触发条件可能是指定的某个时间或日期、特定的用户识别符的出现，特定文件的出现或使用次数等任何因素。如“PETER-2”在每年 2 月 27 日会提出三个问题，答错后会将硬盘加密；著名的“黑色星期五”逢某月 13 日的星期五发作。这些病毒在平时隐藏得很好，只有在条件满足时才会露出本来面目。病毒一般都有一个触发条件，触发条件越多，传染性越强。

7．破坏性

追究计算机病毒的起因：其一是为了表现自己与众不同的编程技能，如 GENP、小球、W-BOOT 等“良性病毒”；其二是为了破坏染毒的计算机系统，使其不能正常运行，如 CIH 病毒。但不论怎样，所有的计算机病毒都是一种可执行程序，而这一可执行程序又必然要运行，因此，所有的计算机病毒都对计算机系统造成不同程度的影响，轻者降低计算机系统的工作效率、占用系统资源（如占用内存空间、磁盘存储空间及系统运行时间等），重者导致数据丢失、格式化磁盘、阻塞网络、系统崩溃。其具体情况取决于计算机病毒设计者的目的和水平。有时几种本没有多大破坏作用的病毒交叉感染，也会导致系统崩溃等重大后果。计算机病毒的破坏性决定了病毒的危害性。

归纳起来，计算机病毒的危害大致有以下几个方面。

1）对计算机数据信息的直接破坏

大部分病毒发作时直接破坏计算机的重要信息数据，攻击系统数据区域，包括：硬盘主引导扇区、引导扇区、FAT 表、文件目录区。攻击的手段有：格式化磁盘、改写文件分配表和目录区、删除重要文件、假冒文件或用无意义的“垃圾”数据改写文件、破坏 CMOS 设置等。例如，磁盘杀手病毒（DISK KILLER），内含计数器，在硬盘染毒后累计开机时间 48 小时内激发，激发的时候屏幕上显示“Warning!! Don’t turn off power or remove diskette while Disk Killer is Prosessing!”（警告！DISK KILLER 在工作，不要关闭电源或取出磁盘），改写硬盘数据。

2）占用磁盘空间和对信息的破坏

寄生在磁盘上的病毒总要非法占用一部分磁盘空间。引导型病毒一般侵占方式是由病毒本身占据磁盘引导扇区，而把原来的引导区转移到其他扇区，也就是引导型病毒要覆盖一个磁盘扇区。被覆盖的扇区数据永久性丢失，无法恢复。

文件型病毒利用一些 DOS 功能进行传染，这些 DOS 功能能够检测出磁盘的未用空间，把病毒的传染部分写到磁盘的未用部位去。所以在传染过程中一般不破坏磁盘上的原有数据，但非法侵占了磁盘空间。一些文件型病毒传染速度很快，在短时间内感染大量文件，每个文件都不同程度地加长了，就造成了磁盘空间的严重浪费。

3）抢占系统资源

大多数病毒在动态下都是常驻内存的，这就必然抢占一部分系统资源。病毒所占用的基本内存长度大致与病毒本身长度相当。病毒抢占内存，导致内存减少，一部分软件

不能运行。除占用内存外，病毒还抢占中断、干扰系统运行。例如，不执行命令、干扰内部命令的执行、虚假报警、打不开文件、内部栈溢出、占用特殊数据区、扰乱串并行口等。

4）影响计算机运行速度

计算机病毒程序为了运行自己的程序，抢占系统资源，进驻内存后不但干扰系统运行，还影响计算机的运行速度。甚至有的病毒在时钟中纳入了时间的循环计数，迫使计算机空转，使计算机的运行速度明显下降。

5）对计算机硬件的破坏

现在一些计算机的硬件很容易遭到计算机病毒的破坏。例如，CIH 病毒侵入主板上的 Flash BIOS，破坏其内容而使主板报废；一些病毒通过篡改显示参数来破坏显示器；一些病毒可以通过修改 BIOS 参数、加高 CPU 电压使其过热而烧坏，或者提高 CPU 的外频，使 CPU 和显卡、内存等因超负荷工作而过热烧坏。

6）给用户造成严重的心理压力

据有关计算机销售部门统计，计算机售后用户因怀疑“计算机有病毒”而提出咨询约占售后服务工作量的 60%以上。经检测确实存在病毒的约占 70%，另有 30%情况只是用户怀疑，而实际上计算机并没有病毒。那么用户怀疑病毒的理由是什么呢？多半是出现诸如计算机死机、软件运行异常等现象。这些现象确实很有可能是计算机病毒造成的，但又不全是。大多数用户对病毒采取“宁可信其有”的态度，这对于保护计算机安全无疑是十分必要的，然而往往要付出时间、金钱等方面的代价。仅仅怀疑病毒而冒然格式化磁盘所带来的损失更是难以弥补。不仅是个人单机用户，在一些大型网络系统中也难免为甄别病毒而停机。总之，计算机病毒像“幽灵”一样笼罩在广大计算机用户心头，给人们造成巨大的心理压力，极大地影响了现代计算机的使用效率，由此带来的无形损失更是难以估量的。

8．针对性

要使计算机病毒得以运行，就必须具有适合其发生作用的特定软硬件环境，即某一种病毒一般只能在某种特定的操作系统和硬件平台上运行，而不可能在所有的操作系统和硬件平台上都能实施攻击功能，这就是计算机病毒的针对性。攻击 UNIX 操作系统的病毒只能对 UNIX 有效，对 DOS、Macintosh 等操作系统不起作用。如 CIH 病毒只能在 Windows 9x 中爆发。计算机病毒的针对性是由计算机病毒的程序性决定的。

9．衍生性

病毒的传染、破坏部分反映了病毒设计者的设计思想和设计目的。但是，这可以被其他掌握原理的人以其个人的企图进行任意改动，从而衍生出另外一种或多种来源于同一种病毒而又不同于原病毒的计算机病毒（又称为变种），这就是计算机病毒的衍生性。衍生性为一些好事者提供了一种创造新病毒的捷径。衍生出来的变种病毒造成的后

果可能比原版病毒更严重。

计算机病毒本身具有的这种衍生性是导致产生变体病毒的必然原因。这种衍生病毒可以是人为结果，也可以是“计算机病毒自动生成”，这种人工智能的结果，是产生多态（也称多形）病毒的温床。

10．寄生性（依附性）

病毒程序嵌入到宿主程序中，依赖于宿主程序的执行而生存，这就是计算机病毒的寄生性。病毒程序在侵入到宿主程序中后，一般对宿主程序进行一定的修改，宿主程序一旦执行，病毒程序就被激活，从而可以进行自我复制和繁衍。

11．不可预见性

从对病毒的检测方面来看，病毒还有不可预见性。不同种类的病毒，它们的代码千差万别，计算机病毒新技术的不断涌现加大了对未知病毒预测的难度，决定了计算机病毒的不可预见性。事实上，反病毒软件预防措施和技术手段往往滞后于病毒的产生。

12．持久性

即使在病毒程序被发现以后，数据和程序乃至操作系统的恢复都非常困难。特别是在网络操作情况下，由于病毒程序由一个受感染的复件通过网络系统反复传播，使病毒程序的清除非常复杂。

10.4.2　计算机病毒的新特征

计算机病毒技术的迅猛发展使计算机病毒出现了与以往迥然不同的特征和发展方向，新病毒出现的频率大为上升，对人类造成的影响越来越大。下面分析计算机病毒的新特征。

1．抗分析性

一些新的病毒采用了加密技术和反跟踪技术。加密技术使病毒分析者无法在不执行病毒的情况下阅读加密的计算机病毒源程序；而反跟踪技术则是使病毒分析者无法动态跟踪计算机病毒程序的运行，从而增加了计算机病毒程序的抗破译能力和伪装能力。例如，Want Job 病毒除了具有自动发信、自动执行及感染局域网等破坏功能外，还能在感染计算机后不断地查询内存中的进程，检查是否有一些诸如 Avp/Nav/Nod/Macfee 等杀毒软件的存在，若存在这些软件，则会将该杀毒软件的进程终止，而且每隔 0.1 s 就循环检查一次，使这些杀毒软件不能运行。

2．诱惑欺骗性

某些病毒不再追求隐蔽性，而是以某种特殊的表现方式引诱、欺骗用户不自觉地触发、激活病毒，从而实施其感染、破坏功能。例如，情书病毒的变种之一 VBS.LoveLetter.F（又名 Virus.Warning），传播该病毒的电子邮件附件名为“Virus_Warning.jpg.vbs”，主题

为“Dangerous virus warning（危险病毒警告）”，其内容是“There is a dangerous virus circulating. Please click attached picture to view it and learn to avoid it（目前病毒正在泛滥，请看附图并学会如何避开病毒）”，引诱用户点击实为 VBS 脚本而非 JPG 图片的附件。2003 年 3 月出现的一种与伊拉克战争有关的 Ganda 病毒，以邮件附件的形式传播。该病毒以许多会对这场战争感兴趣的信息诱惑接收者，并附加上“打电话，行动起来”引诱接收者点击附件，以激活并执行病毒代码。其他迷惑性信息有：“提示用户，无论任何人运行该程序，都会欣赏到一个关于乔治•布什的有趣屏保画面，而且看过之后都不得不承认乔治•布什是一个行动怪异的另类人”等。

3．向多元化发展

传统的病毒攻击对象单一，某一种病毒一般只能在某种特定的操作系统和硬件平台上运行。新出现的病毒可以感染多个操作系统。例如，尼姆达（Worm.Nimda）病毒会感染几乎所有的 Windows 平台。某些病毒既可在 Windows 操作系统下工作，又可以在 Linux 操作系统下工作，目前已经出现了既能感染 Windows PE 文件，又能感染 UNIX/Linux ELF 文件的病毒。病毒向多元化发展的结果是一些病毒更精巧，另一些病毒更复杂，混合多种病毒的特征。例如，红色代码（Code Red）病毒就综合了文件型和蠕虫型病毒的特性。

4．传播方式多样化，传播速度更快

出现了可以通过多种方式进行传播的计算机病毒。这些病毒既可以通过文件传播，又可以通过电子邮件传播；既可以通过局域网快速传播，也可以利用 ISS 4.0/5.0（Internet Security Scanner）的 Unicode 后门进行主动传播。例如，Magistr 变形病毒可以以电子邮件和复制文件两种方式传播。

互联网的广泛普及为计算机病毒的传播打开了方便之门。现在的病毒借助网络的便利，其传播速度已远非原先可比，尤其是一些利用电子邮件系统的病毒。例如，爱虫病毒（VBS.LoveLetter）于 2000 年 5 月 4 日凌晨在菲律宾开始发作，到当天傍晚已经感染了地球另一端的美国的数十万台计算机。

5．攻击技术的混合性

当前流行的病毒采用的攻击技术会将文件传染、蠕虫、黑客等多种技术融为一体。随着病毒技术和黑客技术的发展，病毒编写者最终将这两种技术融合，因此具有这两大特性的病毒会越来越多。例如，Nimda 病毒和 Code Red 病毒的编制者就运用了黑客技术，使病毒的传播和发作无须人为干预。Badtrans 病毒由于运用了黑客的有关技术，能自动地把盗取的个人资料传回给病毒制造者。2003 年 11 月爆发的“爱情后门”（Worm.LoveGate.T）病毒。除了具有蠕虫、黑客、后门等特性外，还具有感染系统文件、盗窃密码等功能。

6．造成的破坏日益严重

CIH 病毒在全球造成的损失估计是 10 亿美元，而受爱虫病毒的影响，全球的损失估计高达 100 亿美元。据来自 Compuware/ABC 的报告，系统每停机一小时，包括证券公司、信用卡公司、电视机构、国际航运公司、邮购公司在内，其损失都在 650 亿美元以上，而对 Internet 公司，尚无人能统计其损失。

10.5　计算机病毒的分类

对计算机病毒分类研究的目的在于更好地描述、分析、理解计算机病毒的特性、危害、原理及其防治技术。尽管计算机病毒的数量非常多，表现形式也多种多样，而且，病毒的数量仍在不断增加，从不同的角度来审视这些病毒，对计算机病毒就会有不同的分类方法。由于同一种病毒可能集多种特征于一身，因此同一种病毒可能会存在多种不同的分类结果。

10.5.1　传统的计算机病毒分类

1．按照计算机病毒攻击的操作系统分类

1）攻击DOS操作系统的病毒

这类病毒出现较早，种类及其变种极多。尽管 DOS 技术在 1995 年以后基本处于停滞状态，但这类病毒的数量和传播仍在发展。例如，PingPong 病毒、Yankee 病毒、DIR II 病毒、One-Half 病毒等。

2）攻击Windows操作系统的病毒

从 1995 年以后，Windows 操作系统逐渐取代 DOS 操作系统而成为微型计算机的主流操作系统，也使其成为病毒的主要攻击对象，攻击 Windows 操作系统的计算机病毒日趋增多。例如，CIH 病毒、Nimda 病毒、Worm.Blaster 病毒等。

3）攻击UNIX操作系统的病毒

随着病毒技术的发展，病毒的攻击目标也开始包括 UNIX 和 Linux。1997 年 2 月，出现了首例攻击 Linux 系统的病毒——Bliss（上天的赐福）病毒；2001 年 4 月，出现了首例在 Windows 和 Linux 操作系统下都能传播的 Win32.Winux 病毒，它主要感染 Windows PE 和 Linux ELF 文件。

4）攻击OS/2 操作系统的病毒

第一个真正属于 OS/2 操作系统意义下的病毒是在 1996 年 2 月发现的 AEP 病毒，该病毒首次能够将自身依附在 OS/2 可执行文件的后面实施感染功能。

5）攻击Macintosh操作系统的病毒

这类病毒的例子出现在 Apple 机上。例如，MacMag（和平）病毒、Scores（评分）

病毒、Elkcloner（埃尔科克隆者）病毒等。

6）其他操作系统上的病毒

如手机病毒、PDA 病毒等。第一例手机病毒是于 2000 年 6 月在西班牙发现的 VBS.Timofonica 病毒。该病毒通过运营商 Telefonica 的移动系统向该系统内的任意用户发送骂人的短消息。

2．按照计算机病毒攻击的机型分类

1）攻击微型计算机的病毒

由于微型计算机及其操作系统、软硬件资源的开放性和应用的广泛性，大多数病毒以微型计算机为攻击对象，因此这类病毒出现得最多、其变种也最多、版本的更新也最快、感染的范围也最为广泛。例如，攻击 Commodore 公司生产的微型计算机的 Amiga 病毒，攻击 Apple 公司生产的微型计算机的 MacMag（和平）病毒，攻击 IBM 和其他公司生产的微型计算机及其兼容机的病毒，如 Brain（大脑）病毒、Stoned（石头）病毒、Marburg（青猴）病毒等。

2）攻击工作站的病毒

计算机硬件的飞速发展使工作站的能力大大加强，并且应用范围也有了较大的发展，所以不难想象，攻击工作站的病毒是对信息系统的一大威胁。

3）攻击小型计算机的病毒

小型计算机的应用极为广泛，既可以作为网络中的节点机，又可以作为网络主机，使小型机也成为了病毒攻击的对象。1988 年 11 月 Morris’s Worm 就是第一例攻击互联网上小型计算机的病毒，改变了病毒只攻击微机的传统观念。又如 WANK.com 和 HL.com 通过 VAX 型号的计算机传播。

4）攻击中、大型计算机的病毒

相对于攻击其他机型的病毒而言，攻击中、大型计算机的病毒很少。尽管如此，对大型计算机的攻击威胁仍然存在。20 世纪 60 年代末，一种大型机——Univax 1108 系统上就发现了一种可将自身链接于其他程序之后的 Pervading Animal（流浪的野兽）病毒。而 20 世纪 90 年代中期，某核电站信息系统的中型计算机遭受到了病毒的侵扰。

5）攻击计算机网络的病毒

在互联网得到空前广泛应用的今天，攻击计算机网络的病毒频繁出现。例如，Morris’s Worm 就是一种网络病毒，又如 Happy99 蠕虫病毒、SirCam（齿轮先生）病毒等。2001 年 7 月出现的 Code Red（红色代码）病毒和同年 8 月出现的 Code Red II 病毒，它们针对互联网上的服务器突施攻击，且能迅速传播，造成网络访问速度的下降乃至阻断。2003 年 8 月出现的 Worm.Blaster（冲击波）蠕虫病毒造成大批计算机瘫痪和网络连接速度变慢。

3．按照计算机病毒的链接方式分类

计算机病毒要进入系统，从而进行感染和破坏，因此，病毒必须与计算机系统内可能被执行的文件建立链接。这些被链接的文件可能是操作系统文件，可能是各种应用程序，还可能是应用程序所用到的数据文件（如 Word 文档）。根据病毒对这些文件的链接形式不同可分为如下几类。

1）源码型病毒

源码型病毒在高级语言（如 C、Fortran、PASCAL 等）编写的程序被编译之前，将病毒代码插入到源程序中，经编译成为合法程序的一部分。这类病毒一般存在于语言处理程序或链接（Link）程序中。第一例感染 C 语言和 PASCAL 语言源代码的病毒是 Srevir 病毒。

2）嵌入型病毒

嵌入型病毒是将自身嵌入到现有程序中，把计算机病毒的主体程序与其攻击的对象以插入的方式链接，使病毒程序与目标程序成为一体。这种计算机病毒是难以编写的，一旦侵入宿主程序体后也较难消除，清除这种病毒时往往会破坏合法程序。这类病毒的数量不多，但破坏力极大，而且很难检测。据报道，1991 年在保加利亚就发现了首例感染 .com 文件的这类病毒。

3）外壳型病毒

外壳型病毒一般链接到宿主程序的首尾，对原来的程序不修改或仅进行简单的修改，相当于给宿主程序加了个“外壳”。当宿主程序执行时首先激活病毒程序。这种病毒最为常见，易于编写，也易于发现，一般测试文件的大小即可察觉。例如，Jerusalem（耶路撒冷）病毒、Dbase（数据库）病毒、Columbus Day（哥伦布日）病毒、Chinese Bomb（中国炸弹）等。

4）操作系统型病毒

操作系统型病毒用自身的程序加入或取代部分操作系统进行工作。这种病毒在运行时用自己的逻辑部分取代操作系统的合法程序模块，从而寄生在计算机磁盘的操作系统区，在启动计算机时，能够先运行病毒程序，然后再运行启动程序。这类病毒具有很强的破坏力，可以导致整个系统的瘫痪，无法启动。例如，Bouncing Ball（小球）病毒、Stoned（石头）病毒（又称大麻病毒）、Pakistan Brain（巴基斯坦）病毒及其变体 Ashar 病毒、Ghost（幽灵）病毒及其变体 Ghost Boot Version 病毒和 Ghost Com Version 病毒、Typo（排字错误）病毒及其变体 Typo Boot Virus 病毒等。

4．按照计算机病毒特有的算法分类

1）伴随型病毒

伴随型病毒并不改变文件本身，它们根据算法产生.EXE 文件的伴随体，具有同样的名字和不同的扩展名（.COM）。例如，XCOPY.EXE 的伴随体是 XCOPY.COM。病毒

把自身写入.COM 文件并不改变.EXE 文件，当 DOS 加载文件时，伴随体被优先执行，再由伴随体加载执行原来的.EXE 文件。

2）“蠕虫”型病毒

“蠕虫”型病毒将计算机网络地址作为感染目标，利用网络从一台计算机的内存传播到其他计算机的内存，将自身通过网络发送。蠕虫通过计算机网络传播，不改变文件和资料信息，除了内存，一般不占用其他资源。

3）寄生型病毒

寄生型病毒依附在系统的引导扇区或文件中，通过系统的功能进行传播。

4）练习型病毒

练习型病毒自身包含错误，不能进行很好的传播，如一些在调试阶段的病毒。

5）诡秘型病毒

诡秘型病毒一般不直接修改中断和扇区数据，而是通过设备技术和文件缓冲区等操作系统内部进行修改，不易看到资源，使用比较高级的技术，利用操作系统空闲的数据区进行工作。

6）变形病毒（又称幽灵病毒）

变形病毒使用一种复杂的算法，一般由一段混有无关指令的解码算法和经过变化过的病毒体组成，使自己每传播一份都具有不同的内容和长度。每一个中毒的文件中所含的病毒码都不一样，更有甚者，几乎无法找到相同的病毒特征码。

5．按照计算机病毒的破坏能力、破坏情况分类

1）根据计算机病毒的破坏能力分类

（1）无害型。除了传染时减少磁盘的可用空间，使可执行程序容量增大外，对计算机系统没有其他影响，如 Elkcloner 病毒。

（2）无危险型。这类病毒仅仅是减少内存、显示些无意义的图像、发出声音等，如女鬼病毒、Bouncing Ball 病毒、MacMag 病毒。

（3）危险型。这类病毒在计算机系统操作中造成严重的错误，如 Hebrew 病毒。

（4）非常危险型。这类病毒删除程序、破坏数据、清除系统内存区和操作系统中重要的信息，如 CIH 病毒。这类病毒对系统造成的危害并不局限于病毒本身的算法中存在的危险调用，还包括当它们传染时引起的无法预料的、灾难性的破坏，如 Klez-H 病毒。

2）按照计算机病毒的破坏情况分类

（1）良性病毒。良性病毒不包含对计算机系统产生直接破坏作用的代码。这类病毒多是出自于恶作剧者之手，编制病毒的目的不是为了对计算机系统进行破坏，而是为了显示其在计算机编程方面的技巧和才华。这类病毒为了表现其存在，只是不停地进行传播，并不破坏计算机系统和数据，但它会使系统资源急剧减少，可用空间越来越少，最

终导致系统崩溃。例如，国内出现的小球病毒就是良性的。良性病毒又可分为无害型病毒和无危险型病毒。

（2）恶性病毒。恶性病毒在代码中包含损伤和破坏计算机系统的操作，在其传染或发作时会对系统产生直接破坏作用。这类病毒的目的就是破坏系统中的信息资源。这类病毒很多，如米开朗基罗病毒，当其发作时，硬盘的前 17 个扇区将被彻底破坏，使整个硬盘上的数据丢失。有的病毒还会对硬盘进行格式化操作。恶性病毒又分为危险型病毒和非常危险型病毒。

需要指出的是，良性和恶性是相比较而言的，不可轻视任何一种病毒对计算机系统造成的危害。

6．按照计算机病毒寄生方式和传染途径分类

计算机病毒按照寄生方式（或传染途径）可分为：引导型病毒、文件型病毒、混合型病毒。

1）引导型病毒

引导型病毒主要用病毒程序取代正常的引导记录，而将正常的引导记录隐藏在磁盘的其他地方。操作系统的引导模块存放在磁盘的某个固定区域（引导区），并且控制权的转接方式是以物理地址为依据的，而不以操作系统引导区的内容为依据，因此，病毒占据了引导区的物理位置即可获得控制权。这种病毒在运行的一开始（如系统启动）就能获得控制权，其传染性较大。它们可感染启动扇区（Boot），如 Brain 病毒和小球病毒；感染主引导扇区（Master Boot Record，MBR），如 Stoned（大麻）病毒等。

2）文件型病毒

文件型病毒通过操作系统的文件系统实施感染，它们以感染可执行文件为主，如 BAT、EXE、COM、SYS、DLL、OVL、VXD 文件等。也有一些可以感染其他文件，如 OBJ、DOC、DOT 文件等。文件型病毒的安装必须要借助病毒的载体程序，通过运行病毒的载体程序将文件型病毒引入内存，伺机感染其他文件或直接感染其他文件。已感染病毒的文件执行速度会减慢，甚至无法运行。例如，感染 OBJ 文件的 shifter（移动者）病毒、感染 Windows 平台下 Java 可执行文件的 Strange Brew（奇异的酿酒）病毒、CIH 病毒等。

目录型病毒是文件型病毒的特例，其感染方式非常独特，仅通过修改目录区便可以达到感染的目的，如 DIRII 病毒。

尽管宏病毒是一种新形态的计算机病毒，但宏病毒的一个基本特征是隐藏在字处理文件或电子数据表中，感染 DOC 文档和 DOT 模板文件，对用户系统中的可执行文件和数据类文件造成破坏，打破了“数据文件不会传播病毒”的神话。因此宏病毒从本质上讲是同类文件型病毒。后面还会单独介绍。

3）混合型病毒

混合型病毒有时也称多型病毒，它综合了引导型病毒和文件型病毒的特点，同时能够感染文件和引导扇区两种目标。这类病毒通常都具有复杂的算法，使用非常规的方法攻击计算机系统，同时使用了加密和变形算法。例如，1991 年出现的 Tequila（蒸馏酒）病毒、Amoeba（变形虫）病毒等。

7．按计算机病毒传播媒介分类

按照计算机病毒的传播媒介来分，病毒可分为单机病毒和网络病毒。

1）单机病毒

单机病毒的载体是磁盘、优盘、光盘等移动存储设备。常见的是病毒从软盘传入硬盘，感染系统，然后再传染其他软盘，软盘又传染其他系统。

2）网络病毒

网络病毒的传播媒介不再是移动式载体，而是网络通道，利用计算机网络的协议或命令及 E-mail 等进行传播，常见的是通过 QQ、BBS、E-mail、FTP、Web 等传播。这种病毒的传染能力、破坏力更强。

10.5.2　恶意程序、宏病毒、脚本病毒、蠕虫、特洛伊木马

1．恶意程序

恶意程序是一个概括性的术语，指任何故意创建用来执行未经授权并通常带有攻击意图的一段程序。后门程序、逻辑炸弹、特洛伊木马、蠕虫、细菌、病毒、犯罪软件、间谍软件等都是一些可以称之为恶意程序的例子。恶意程序这一术语的产生有其必然性。在过去，主要由基于 DOS 的计算机组成的环境中，称某些文件为“病毒”或“木马”就足够了，因为那时病毒种类还很有限。当时也有“蠕虫”。然而随着感染方式和病毒的不断演变，术语“病毒”和“木马”已经不再是对存在的各种恶意程序的令人满意的定义。

恶意程序大致可以分成两个类别：需要宿主程序的恶意程序和彼此独立的恶意程序。前者基本上不能独立于某个实际的应用程序或系统程序，即存在宿主，这种恶意代码必须依赖宿主的运行而启动；后者是可以被操作系统调度和运行的独立的程序。也可以将恶意程序分成不进行复制工作和进行复制工作的。简单地说，前者是当宿主程序调用时被激活起来完成一个特定功能；后者或者由程序片段（病毒）或者由独立程序（蠕虫、细菌）组成，在执行时可以在同一个系统或某个其他系统中产生自身的一个或多个以后被激活的副本。

2．宏病毒

因为宏病毒（Macro Virus）与传统的病毒有很大的不同，因此将宏病毒单独列出来介绍。它不感染 EXE、COM 等可执行文件，而是将病毒代码以宏（Macro）的形式潜伏

在 Office 文档中，当打开这些染毒文件时，这些代码就会被执行并产生破坏作用。

在了解宏病毒之前，需要了解什么是宏。宏，译自英文单词 Macro，是微软公司为其 Office 软件包设计的一个特殊功能，软件设计者为了让人们在使用软件进行工作时，避免一再地重复相同的动作而设计出来的一种工具，它利用简单的语法，把常用的动作写成宏，在工作时，就可以直接利用事先编好的宏自动运行，目的是让用户文档中的一些任务自动化。如果一个宏中包含了有破坏能力的命令，并且还有自我复制功能，这个宏就成了宏病毒。

由于在目前发现的宏病毒当中，感染 Word 的占绝大多数，所以人们谈论的宏病毒一般是指 Word 宏病毒。Word 提供了两种创建宏的方法：宏录制器和 Visual Basic 编辑器。在默认的情况下，Word 将宏存储在通用模板（Normal.dot）中，以便所有的 Word 文档均能使用，这一特点几乎为所有的宏病毒所利用。如果撰写了有问题的宏，感染了通用模板，那么只要一执行 Word，这个受感染的通用模板便会传播到之后所编辑的文档中去，如果其他用户打开了感染病毒的文档，宏病毒又会转移到他的计算机上。这就是日常所说的寄存在文档或模板宏中的宏病毒。除了 Word 宏病毒外，常见的还有 Excel 宏病毒、PowerPoint 宏病毒等，主要是针对 Office 软件的。

宏病毒的主要类型：有些宏病毒对用户进行骚扰，但不破坏系统。例如，有一种宏病毒在每月的 13 日发作时显示出 5 个数字连乘的心算数学题。有些宏病毒使打印中断或打印出混乱信息，如 Nuclear、Kompu 等属此类。有些宏病毒将文档中的部分字符、文本进行替换。但也有些宏病毒极具破坏性，如 Mdma.a，这种病毒既感染中文版 Word，又感染英文版 Word，发作时间是每月的 1 日。此病毒在不同的 Windows 平台上有不同的破坏性表现，轻则删除帮助文件，重则删除硬盘中的所有文件。另外，还有一种双栖复合型宏病毒，发作可使计算机瘫痪。

随着 Office 软件在全世界范围内的不断普及，到 1998 年宏病毒已经成为发展最快、传播最迅速、危害最大的一类病毒。美国国际计算机安全协会 ICSA 的《ICSA 1998 病毒流行调查报告》表明宏病毒占 1998 年十大病毒感染事件的 72%，在十大病毒中占了五席交椅：WM/Concept、WM/Cap、WM/Wazzu、WM/Npad 和 XM/Laroux。根据 DataFellow 公司每天特征码升级的 F-Macro 所检测到的宏病毒数目，至 1998 年 12 月达到了 3 332 个，而在 1997 年 12 月才是 1 821 个，增长率为 83.0%。

3．脚本病毒

脚本病毒（Script Virus）的前缀是：Script。脚本病毒的共同特性是使用脚本语言编写，通过网页进行传播。在 1999 年以后成了主要的病毒类型之一，如 LoverLetter 病毒。脚本病毒类似于宏病毒，但它的执行环境不再局限于 Office 应用程序，而是随着微软将脚本语言（Script Language）和视窗操作系统日益紧密结合，扩展到网页、HTA，甚至文本文件中。

脚本病毒会有如下前缀：VBS、JS，如欢乐时光（VBS.Happytime）、十四日（Js.Fortnight.c.s）等。VBS 病毒是用 VB Script 编写而成，JS 病毒是用 Java Script 编写而成。还有基于 PHP 的脚本病毒、脚本语言和木马结合的病毒。VBS 病毒的脚本语言功能非常强大，它们利用 Windows 系统的开放性特点，通过调用一些现成的 Windows 对象、组件，可以直接对文件系统、注册表等进行控制。以爱虫和新欢乐时光病毒为典型代表的 VBS 脚本病毒十分猖獗，很重要的一个原因就是其编写简单。下面介绍 VBS 脚本病毒的一些特点。

（1）编写简单，一个以前对病毒一无所知的病毒爱好者可以在很短的时间里编出一个新型病毒。

（2）破坏力大。其破坏力不仅表现在对用户系统文件及性能的破坏，还可以使邮件服务器崩溃，使网络发生严重阻塞。

（3）感染力强。由于脚本是直接解释执行的，并且它不需要像 PE 病毒那样需要做复杂的 PE 文件格式处理，因此这类病毒可以直接通过自我复制的方式感染其他同类文件，自我的异常处理变得非常容易。

（4）传播范围大。这类病毒通过 htm 文档、E-mail 附件或其他方式，可以在很短时间内传遍世界各地。

（5）病毒源码容易被获取，变种多。由于 VBS 病毒解释执行，其源代码可读性非常强，即使病毒源码经过加密处理后，其源代码的获取还是比较简单。因此，这类病毒变种比较多，稍微改变一下病毒的结构或修改一下特征值，很多杀毒软件可能就无能为力了。

（6）欺骗性强。脚本病毒为了得到运行机会，往往会采用各种让用户不注意的手段。例如，邮件的附件名采用双后缀，如.jpg.vbs，由于系统默认不显示后缀，这样，用户看到这个文件的时候，就会认为它是一个 jpg 图片文件。

（7）使病毒生产机实现起来非常容易。所谓病毒生产机，就是可以按照用户的意愿生产病毒的机器（这里指的是程序）。目前的病毒生产机之所以大多数都为脚本病毒生产机，其中最重要的一点还是因为脚本是解释执行的，实现起来非常容易。

正因为以上几个特点，脚本病毒发展异常迅猛，特别是病毒生产机的出现，使生成新型脚本病毒变得非常容易。

4．蠕虫

1）蠕虫的定义

蠕虫是一段程序。最初的蠕虫定义是因为在 DOS 环境下，发作时会在屏幕上出现一条类似虫子的东西，胡乱吞吃屏幕上的字母并将其变形。

从广义上定义，凡能够引起计算机故障、破坏计算机数据的程序统称为计算机病毒。所以从这个意义上说，蠕虫（Worm）也是一种病毒！但是蠕虫和一般的病毒有着

很大的区别。1988 年 Morris（莫里斯）蠕虫爆发后，Eugene H.Spafford 为了区分蠕虫和病毒，给出了蠕虫的技术角度定义：“Worm is a program that can run by itself and can propagate a fully working version of itself to other machines.”即蠕虫可以独立运行，并能把自身的一个包含所有功能的版本传播到另外的计算机上。将病毒的含义进一步解释为：“Virus is a piece of code that adds itself to other program, including operating systems. It cannot run independently and it requires that its ‘host’ program be run to activate it.”即病毒是一段代码，能把自身加到其他程序包括操作系统上。它不能独立运行，需要由它的宿主程序来激活它。

蠕虫和病毒具有一些共性，如传播性、隐蔽性、破坏性和复制功能，这些主要特征上的一致导致二者难以区分，尤其是近年来，越来越多的病毒采用了部分蠕虫的技术，另外，蠕虫也采用了部分病毒的技术，更加剧了这种情况。但蠕虫具有自己的一些特征，在蠕虫的定义中强调了自身副本的完整性和独立性。蠕虫不需要将自身链入宿主程序来达到自我复制的目的。蠕虫传播无须用户操作，主要利用计算机系统漏洞（Vulnerability）进行传染，搜索网络中存在漏洞的计算机后主动攻击，可以对网络造成拒绝服务。在产生的破坏性上，蠕虫也不是普通病毒所能比拟的，网络的发展使蠕虫可以在很短的时间内蔓延整个网络，造成网络瘫痪。

根据使用者的情况可将蠕虫分为两类：一类是面向企业用户和局域网而言的，这种蠕虫利用系统漏洞，主动进行攻击，可以对整个互联网可造成瘫痪性的后果。以“红色代码”、“尼姆达”及最新的“SQL 蠕虫王”为代表；另外一类是针对个人用户的，通过网络（主要是电子邮件、恶意网页形式）迅速传播，以爱虫、求职信为代表。在这两类中，第一类具有很大的主动攻击性，而且爆发也有一定的突然性，但相对来说，查杀这种病毒并不是很难的。第二类的传播方式比较复杂和多样，少数利用了微软应用程序的漏洞，更多地利用对用户进行欺骗和诱使，这样的病毒造成的损失是非常大的，同时也是很难根除的，如求职信蠕虫，在 2001 年就已经被各大杀毒厂商发现，但直到 2002 年年底依然排在病毒危害排行榜的首位，这就是证明！

2）蠕虫的破坏

1988 年莫里斯蠕虫开始现身网络；后来的红色代码、尼姆达蠕虫疯狂时，造成几十亿美元的损失；北京时间 2003 年 1 月 26 日，一种名为“2003 蠕虫王”的计算机蠕虫迅速传播并袭击了全球，致使互联网网络严重堵塞，作为互联网主要基础的域名服务器（DNS）的瘫痪造成网民浏览互联网网页及收发电子邮件的速度大幅减缓，同时银行自动提款机的运作中断，机票等网络预订系统的运行中断，信用卡等收付款系统出现故障。专家估计，此蠕虫造成的直接经济损失至少在 12 亿美元以上！蠕虫的危害如表 10-1 所示。

表 10-1　蠕虫的危害

蠕 虫 名 称	持 续 时 间	造 成 损 失
莫里斯	1988 年	6000 多台计算机停机，直接经济损失达 9600 万美元
美丽杀手	1999 年	政府部门和一些大公司紧急关闭了网络服务器，经济损失超过 12 亿美元
爱虫	2000 年 5 月至今	众多用户计算机被感染，损失超过 100 亿美元
红色代码	2001 年 7 月	网络瘫痪，直接经济损失超过 26 亿美元
求职信	2001 年 12 月至今	大量病毒邮件堵塞服务器，损失达数百亿美元
蠕虫王	2003 年 1 月	网络大面积瘫痪，银行自动提款机运作中断，直接经济损失超过 26 亿美元

由表 10-1 可以知道，蠕虫病毒对网络产生堵塞作用，并造成了巨大的经济损失!

3）蠕虫的特点和发展趋势

（1）利用操作系统和应用程序的漏洞主动进行攻击。此类蠕虫主要是“红色代码”和“尼姆亚”及至今依然肆虐的“求职信”等。由于 IE 浏览器的漏洞（IFRAME EXECCOMMAND），使感染了“尼姆亚”的邮件在不去手工打开附件的情况下病毒就能被激活，而此前即便是很多防病毒专家也一直认为：带有病毒附件的邮件，只要不去打开附件，病毒不会有危害。“红色代码”是利用了微软 IIS 服务器软件的漏洞（idq.dll 远程缓存区溢出）来传播的，“SQL 蠕虫王”则利用了微软的数据库系统的一个漏洞进行大肆攻击。

（2）传播方式多样。蠕虫入侵网络的主要途径是通过工作站传播到服务器硬盘中，再由服务器硬盘的共享目录传播到其他的工作站，其传播的方式比较复杂。如“尼姆亚”和”求职信”，可利用的传播途径包括文件、电子邮件、Web 服务器、网络共享等。

（3）制作技术新。与传统的病毒不同的是，许多新蠕虫是利用当前最新的编程语言与编程技术实现的，易于修改以产生新的变种，从而逃避反病毒软件的搜索。另外，它们还利用 Java、ActiveX、VB Script 等技术，可以潜伏在 HTML 页面里，在上网浏览时触发。

（4）与黑客技术相结合，潜在的威胁和损失更大。以红色代码为例，感染后机器的 Web 目录的\scripts 下将生成一个 root.exe，可以远程执行任何命令，从而使黑客能够再次进入。

（5）对个人用户产生直接威胁。对于个人用户而言，威胁大的蠕虫采取的传播方式一般为电子邮件（E-mail）和恶意网页等。对于利用电子邮件传播的蠕虫来说，通常利用的是各种各样的欺骗手段诱惑用户点击的方式进行传播。恶意网页确切地讲是一段黑客破坏代码程序，它内嵌在网页中，当用户在不知情的情况下打开含有病毒的网页时，病毒就会发作。这种病毒代码镶嵌技术的原理并不复杂，所以会被很多怀有不良企图者利用，在很多黑客网站竟然出现了关于用网页进行破坏的技术论坛，并提供破坏程序代码下载，从而造成了恶意网页的大面积泛滥，也使越来越多的用户遭受损失。

5．特洛伊木马

特洛伊木马（Trojan Horse），简称木马，它得名于古希腊的一个神话传说，特洛伊木马表面上是“礼物”，但实际上是藏匿了袭击特洛伊城的希腊士兵。现在，特洛伊木马是一些表面有用的软件程序，实际目的是危害计算机安全并破坏计算机系统。它定义为一种与远程计算机之间建立起连接，使远程计算机能够通过网络控制用户计算机系统，并且可能造成用户的信息损失、系统损坏甚至瘫痪的非法程序。

通常木马并不被当做病毒，因为它们通常不包括感染程序，因而不能自我复制，但是木马和病毒一样具有破坏性。一般的木马都有客户端和服务器端两个执行程序，其中客户端用于攻击者远程控制植入木马的计算机的程序，服务器端是在用户计算机中的木马程序。攻击者要做的第一步就是把木马的服务器端程序植入到用户的计算机中。木马入侵的主要途径是通过邮件和下载。例如，邮件的附件声称是 Microsoft 安全更新程序，但实际上是一些试图禁用防病毒软件和防火墙软件的木马。一旦用户禁不起诱惑打开了自认为合法的程序，木马便趁机传播。另外，木马也可能包含在免费下载软件中，木马可执行文件一般非常小，如果把木马捆绑到其他正常文件上，用户很难发现，有一些网站提供的下载软件往往捆绑了木马文件，当用户执行这些下载文件的同时，也运行了木马。木马也可以通过 Script、ActiveX、Asp 及 CGI 交互脚本的方式植入。

木马植入被攻击的主机后，一般会通过一定的方式把入侵主机的信息，如主机的 IP 地址、木马端口等发送给攻击者，攻击者有了这些信息就能够和木马里应外合，进行控制被攻击的主机。

流行木马有如下基本特征。

1）隐蔽性

隐蔽性是木马首要的特征。为了防止木马被发现，木马的设计者会采用各种手段隐藏木马，使木马在用户不易察觉的情况下完成一些危害用户的操作。这也是木马和远程控制软件的最主要区别。木马服务器端在运行的时候不产生图标，不出现在任务管理器中。

2）自动运行性

木马在被入侵主机的启动配置文件（如 win.ini、system.ini、winstart.bat 等）、启动组或注册表中写入自己的启动项。在被入侵主机系统启动时跟随启动。

3）欺骗性

木马为了隐藏自己，就要借助系统中已有的文件。名字方式：木马经常使用类似于常见的文件名或扩展名（如 dll、win、sys）的名字，或者用一些不易被人区分的文件名（如字母“l”与数字“1”、字母“o”与数字“0”）。相同文件名但不同路径：借用系统文件中已有的文件名，只不过保存在不同的路径下。常用图标 ZIP：把自己设置成一个 ZIP 文件式图标，当用户不小心打开它时，它就马上运行。

4）自动恢复功能

现在很多木马程序中的功能模块已不再是单一的文件，而是具有多重备份，将文件分别存储在不同的地方。这些分散的文件可以相互恢复，以提高存活能力。

5）自动打开特别的端口

当被木马入侵的主机连网时，木马就会用服务器客户端的通信手段把信息告诉攻击者，以便攻击者控制用户的计算机，或者实施进一步的攻击。根据 TCP/IP 协议，每台计算机可以有 256*256 个端口，即从 0～65535 号端口，其中只有少数几个是常用的，木马利用不常用的端口进行连接。

6）功能的特殊性

木马的功能都是十分特殊的，除了普通的文件操作外，还有搜索缓存中的口令、设置口令、扫描目标机器的 IP 地址、进行键盘记录、远程注册表的操作及锁定鼠标等功能。而这些功能是通常的远程控制软件不具有的。

7）黑客组织趋于公开化

以往没有公开化的病毒组织，因为病毒制作者一般不敢公开发布木马，一旦被发现，是会被判刑或罚款的。现今，专门开发木马程序的组织到处都是，而且还公开在网上出售木马程序，似乎这些组织已经合法化了。

如果按木马程序的破坏方式分类，目前的木马程序可以分为以下几类：远程控制型木马、发送密码型木马、键盘记录型木马、破坏型木马、FTP 型木马。

10.6　计算机病毒的传播

计算机病毒具有自我复制和传播的特点，因此了解计算机病毒的传播途径是极为重要的，目前主要有以下四种途径。

（1）通过不可移动的计算机硬件设备进行传播。这些设备通常有计算机的专用 ASIC 芯片和硬盘等。

（2）通过移动存储设备来传播。这些设备包括软盘、磁盘、优盘等。这些使用最广泛、移动最频繁的存储介质也成了计算机病毒寄生的“温床”。

（3）通过 Internet（计算机网络）进行传播。网络是人们生活中不可缺少的一部分，越来越多的人利用它来获取信息、发送和接收文件、接收和发布新的消息及下载文件和程序，正因为如此，计算机病毒也走上了高速传播之路。在信息国际化的同时，病毒也在国际化，这种方式已成为病毒的最主要的传播途径。

- 病毒以附件的形式通过电子邮件进行传播。使电子邮件成为当今世界上传播计算机病毒最主要的媒介。
- 通过WWW浏览。目前 互联网上有些别有用心的人利用Java Applets和ActiveX Control来编写计算机病毒和恶性攻击程序，因此WWW浏览感染计算机病毒的

可能性也在不断增加。

- 利用 BBS 松散的安全管理，使之成为计算机病毒传播的场所。
- 通过 FTP（文件传输协议）文件下载，使互联网上的病毒传播更容易、更广泛，这一途径能传播现有的所有病毒。
- 利用接收新闻组，这些信息中包含的附件有可能使用户的计算机感染计算机病毒。
- 通过 IM（即时通信系统，如 QQ、MSN），像电子邮件一样，消息服务同样可以自由地传播文件，从而成为病毒传播的主要途径之一。

（4）无线通道传播。目前，该传播途径还不是十分广泛，但预计在未来的信息时代，无线通道途径很可能与网络传播途径成为病毒扩散的两大“时尚渠道”。

10.7　计算机病毒的结构和工作机理

计算机病毒是一段特殊的程序，尽管目前出现的病毒数量及种类繁多，但它们的程序结构都存在着许多共同之处。绝大多数计算机病毒都是由引导模块（主控模块）、传染模块和破坏模块（表现模块）这 3 个基本的功能模块所组成的。其中后两个模块都包含一段触发条件检查程序段，它们分别检查是否满足传染条件及是否满足破坏（表现）条件，一旦相应的条件得到满足，病毒就会进行传染和破坏（表现）。计算机病毒的程序结构如表 10-2 所示。

表 10-2　计算机病毒的程序结构

引 导 模 块	
传染模块	激活传染条件判断部分
	传染功能实施部分
破坏（表现）模块	触发破坏（表现）条件判断部分
	破坏（表现）功能实施部分

引导模块的功能是使病毒获得执行并使其后面的两个模块处于激活状态。传染模块的功能是在传染条件满足时把病毒传染到所攻击的对象上。破坏（表现）模块的功能是在病毒破坏、表现条件（发作条件）满足时，实施对系统的干扰和破坏活动。计算机病毒的一般工作流程用类 C 语言描述如下：

```
/*引导功能模块*/
{将病毒程序寄生于宿主程序中；
加载计算机程序；
病毒程序随其宿主程序的运行进入系统；}
{传染功能模块；}
{破坏功能模块；}
```

```
Main()
{调用引导功能模块;
A：do
{寻找传染对象;
if（传染条件不满足）
goto A;}
while（满足传染条件）
调用传染模块;
while（满足破坏条件）
{激活病毒程序;
调用破坏功能模块；}
运行宿主源程序;
if 不关机
goto A;
关机;
}
```

例 10.1　下面给出一个用 Pascal 语言编写的非常简单的良性病毒的程序作为例子（行号是为了方便而加的）。

```
Program Virus_command;                    {寄生于 COMMAND.COM 中的病毒}      (1)
Uses Crt,dos;                                                                (2)
Var First3:strint[3];User_Command:string;                                    (3)
Begin                                                                        (4)
Repeat                                                                       (5)
    Write('A>');                          {给出一个正常的假象}               (6)
    Readin(User_Command);                 {读入用户命令}                     (7)
    Exec('A:\Comm.com','/C'+User_Command);    {用原解释器解释执行用户命令}   (8)
    First3:=copy(User Command,1,3);           {查命令前三字符}               (9)
    if(First3='dir')or(First3:='ver')then {检查条件，看时机是否成熟}         (10)
    Begin                                 {命令为 dir 或 ver，时机成熟}      (11)
    User_Command:='Copy A:\Comm.com'+'B:\Comm.com';  {恢复原解释器到 B:}     (12)
    Exec('A:\Comm.com','/C'+User_Command);                                   (13)
    User_Command:='Copy A:\Command.com'+'B:\Command.com'; {复制自身到 B}     (14)
    Exec('A:\Comm.com','/C'+User_Command);                                   (15)
    Writein("This is a virus program");                {表现自己}            (16)
    End;
    Until False;
End
```

用已感染此病毒的 A 盘（系统盘）启动机器后，COMMAND.COM（COMMAND.COM 为病毒文件本身，原 COMMAND.COM 文件被改名为 COMM.COM）首先被执行，从而引导了寄生的病毒。所以此病毒的引导机制是利用了 DOS 系统的引导机制，用一个假的 COMMAND.COM 来欺骗系统。执行到第 6 行将显示 A> 来欺骗用户，等候用户输入命令 User_Command，第 8 行正常执行用户命令，第 9～10 行等候时机，每当用户输入命令为“dir”或“ver”时，开始繁殖，这是病毒的传染模块（包括传染条件判断部分和实施传染部分）。最后显示出“This is a virus program”，是病毒的表现模块。

在第 8 行调用了真正的命令解释器 COMM.COM 来解释执行用户的命令，因此程序能给人以平安无事的假象，而实际对原 COMMAND.COM 进行了盗用。此病毒可以看成以外壳的形式寄生在原 COMMAND.COM（现改为 COMM.COM）外，是一个外壳式病毒。

需要说明的是：并非所有的病毒程序都全部具备如表 10-2 所示的 3 个基本模块。例如，文件类型的 Vienna 病毒只有传染模块和破坏模块而没有引导模块。这类病毒利用操作系统的加载机制瞬间动态执行传染和破坏模块，瞬时轰炸，打了就跑。而引导类型的 Brain 病毒只有引导模块、传染模块而无表现模块，这正是某些病毒不具有“表现性”的原因所在。但是，没有表现性却增强了这类病毒的隐蔽性。而有一些病毒在 3 个模块之间可能没有明显的界线。另外，病毒一般还都有自己的潜伏（或隐藏）技巧。

下面对病毒的几种工作机理进行简要介绍。

10.7.1　病毒的引导机理

计算机病毒实际上是一种特殊的程序。该程序必然要存储在某种介质上，为了进行自身的主动传播，病毒程序必须寄生在可以获得执行权的对象上。就目前出现的各种计算机病毒而言，其寄生的对象有两种：寄生在磁盘引导扇区和寄生在特定文件（EXE 和 COM 等可执行文件上、DOC 和 HTML 等非执行文件）中。

计算机病毒寄生的目的就是找机会执行引导模块。从而使自己处于活动状态。计算机病毒的引导过程一般包括以下三个方面：驻留内存、窃取系统控制权、恢复系统功能。

对于寄生在磁盘引导扇区中的病毒来说，病毒引导程序占有了原来系统引导程序的位置，并把原系统引导程序搬移到一个特定的地方。这样系统一启动，病毒引导模块就会自动地装入内存并获得执行权，然后该引导程序负责将病毒的传染模块和破坏（表现）模块装入内存的适当位置，并采取常驻内存技术以保证这两个模块不会被覆盖，接着对这两个模块设定某种激活方式，使之在适当的时候获得执行权。完成这些工作后，病毒引导模块将原系统引导程序装入内存，使系统在带毒状态下依然可以继续运行。

对于寄生在文件中的病毒来说，病毒程序一般通过修改原有文件使对该文件的操作转入病毒程序引导模块，引导模块也完成把病毒的其他两个模块驻留内存及初始化的工作，然后把执行权交给原文件，使系统及文件在带毒的状态下继续运行。

2001 年 4 月前后出现的 I-Worm.Magister 病毒是一个文件类型病毒。该病毒实现驻留内存的方法是：取得对 Explorer.exe 进程内存的控制，通过 110 个字节左右的加载例程，将该病毒程序代码在 Explorer 的内存空间中运行。

10.7.2　病毒的传染机理

传染是病毒最本质的特征之一，是病毒的再生机制。所谓传染是指计算机病毒由一个载体传播到另一个载体、由一个系统进入另一个系统的过程。

计算机病毒的传染分为被动传染和主动传染。对于被动传染，其传染过程是随着复制和网络传输工作的进行而进行的。对于主动传染，其传染过程是在系统运行时，病毒通过系统的外存储器进入到系统的内存储器，然后，常驻内存并监视系统的运行，在一定条件下采用多种手段进行传染。

计算机病毒的传染方式基本上可以分为两大类：一是立即感染，即病毒在被执行的瞬间，抢在宿主程序开始执行前，立即感染磁盘上的其他程序，然后再执行宿主程序；二是驻留内存并伺机传染，内存中的病毒检查当前系统环境，在执行一个程序、浏览一个网页时传染磁盘上的程序。驻留在系统内存中的病毒程序在宿主程序运行结束后，仍可运行，直至关闭计算机。

当执行或使用被感染的文件时，病毒就会加载到内存，接着病毒便开始监视系统的运行，当它发现被传染的目标时，进行如下操作:

（1）根据病毒自己的特定标识来判断文件是否已感染该病毒；

（2）当条件满足时，将病毒链接到文件的特定部位，并存入磁盘中；

（3）完成传染后，继续监视系统的运行，试图寻找新的攻击目标。

例如，CIH 病毒只针对 Windows 95/98 操作系统环境下 PE 格式的可执行文件，要判断这类文件是否已经被病毒感染过。有些病毒程序寄生在可执行文件的头部或尾部，只要运行这个可执行文件，首先执行的是病毒程序，将病毒常驻内存，再设置触发条件。然后转移到正常程序部分，执行正常程序。

10.7.3　病毒的破坏（表现）机理

病毒程序的引导模块和传染模块是为破坏（表现）模块服务的。破坏模块在设计原则、工作原理上与传染模块基本相同。一般情况下，病毒是基于一个或若干个设定的破坏条件都满足的情况下才触发破坏（表现）功能。

计算机病毒的破坏行为体现了病毒的杀伤力。病毒破坏行为的激烈程度取决于病毒作者的主观愿望和他所具有的技术水平。病毒破坏行为千奇百怪，破坏的目标主要有：系统数据区、文件、内存、系统运行速度、磁盘、CMOS、主板和网络等。

10.7.4　病毒的触发机理

可触发性是病毒的基本特性，是病毒的破坏性和潜伏性之间的调整杠杆。可以控制病毒感染和破坏的频率。

计算机病毒在传染和破坏之前，往往要判断某些特定的条件是否满足，满足则传染或发作。这个条件就是计算机病毒的触发条件。实际上，计算机病毒采用的触发条件花样繁多，归纳起来有如下几种。

（1）日期触发：许多病毒采用日期作为触发条件。日期触发大体包括特定日期触发、月份触发、前半年/后半年触发等。例如，“黑色星期五”病毒的触发条件就是日期，逢某月 13 日的星期五发作。CIH 病毒于每年的 4 月 26 日发作。

（2）时间触发：时间触发包括特定时间触发、染毒后累计工作时间触发、文件最后写入时间触发等。例如，Yankee Doodle（杨基歌）病毒以时间为触发方式，病毒常驻内存以后，每当 17:00（下午 5 时）便演奏杨基歌这首曲子。Disk Killer（磁盘杀手）病毒对 PC 每秒 13.2 次的时钟中断进行计数，当计数值达到 300 000H 时，病毒被触发。这时键盘死锁，一些杂乱的数据被写入硬盘，直至全部数据被毁坏为止。

（3）键盘触发：有些病毒监视用户的击键动作，当发现病毒预定的击键时，病毒被激活，进行某些特定的操作。键盘触发包括击键次数触发、组合键触发、热启动触发等。例如，产于墨西哥的 Devil’s Dance（魔鬼之舞）病毒，监视键盘，在 2000 次按键时，改变显示器颜色，在 5000 次按键时，破坏硬盘 FAT 表。产于中国台湾的 Invader（入侵者）病毒，当计算机热启动（按下 Ctrl+Alt+Del 组合键）时，病毒发作，系统硬盘第一磁道的数据被破坏。

（4）感染触发：许多病毒的感染需要某些条件触发，而有相当数量的病毒又以与感染有关的信息反过来作为其破坏行为的触发条件，称为感染触发。包括运行感染文件个数触发、感染序数触发、感染磁盘数触发和感染失败触发。感染序数触发是对自身感染的文件个数计数，而运行感染文件个数触发是既对自身感染的文件计数，又对其他病毒程序感染过的文件计数。例如，“黑色星期一”病毒便是运行感染文件个数触发，该病毒于 1990 年首先在马来西亚发现，对运行的病毒程序进行计数，当计算机运行第 240 个染毒程序时，病毒发作，格式化硬盘。Yankee Doodle（杨基歌）病毒是感染序数触发，当感染 100 次时，病毒代码自毁，一个文件中的病毒代码不治而愈，文件突然短一截，也会给人以惊扰。

（5）启动触发：病毒对计算机的启动次数计数，并将此值作为触发条件。1991 年在西班牙发现的 Anti-Tel 病毒，对计算机启动次数计数，在 400 次启动时病毒发作，将硬盘中的文件全部破坏，并显示信息：“VIRUS ANTIRELEFONICA(BARCELONA)”。

（6）访问磁盘次数触发：病毒对磁盘 I/O 访问次数进行计数，以预定的次数作为触发的条件。例如，1989 年 11 月最先在印度发现的 Print Screen 病毒，该病毒的变种 Print

Screen-2 病毒对磁盘 I/O 操作进行计数，当第 255 次磁盘 I/O 操作时，病毒将屏幕内容送打印机打印。

（7）调用中断功能触发：这类病毒对中断功能调用次数计数，以预定次数作为触发条件。例如，1991 年 9 月在南非发现的 Poem 病毒，该病毒常驻内存后，对中断 INT 21H 的调用次数进行计数，当计数到 2112 次时，屏幕显示病毒信息。

（8）CPU 型号/主板型号触发：病毒能识别运行环境的 CPU 型号/主板型号，以预定 CPU/主板型号作为触发条件。这种病毒的触发方式奇特罕见。例如，1990 年 12 月首先在美国发现的 Violator B4 病毒，这是一个非常特殊的病毒，其行为取决于运行的 PC 采用什么 CPU。如果是 8088 CPU 系统，该病毒除了感染不做任何其他动作。如果发现使用 80286 及 80286 以上的 CPU 处理器，立即激活病毒，它将系统硬盘的开头部分覆盖掉，并显示有关圣诞节问候的信息。

（9）打开邮件触发：这类病毒一般是借助电子邮件这个通信工具来传播的，当用户预览邮件或运行附件时就满足了该病毒的触发条件。例如，尼姆达（Nimda）蠕虫，只要一浏览邮件立刻中毒，同时读取用户信件，取出 SMTP 地址、邮箱地址，利用这些地址将带有蠕虫的信件向外发送，而且在局域网内传播。

（10）随机触发：这些病毒的发作并不需要固定的触发条件，而是随机进行破坏活动。例如，Girlhost（女鬼）病毒，随着贺岁邮件到处传播。“女鬼”病毒溜进计算机时没有任何征兆，并且发作也是随机的。它在发作时会先讲述一段关于“女鬼”的凄美故事，然后突然出现让人毛骨悚然的画面和声音。

（11）利用系统或工具软件的漏洞触发：这类病毒专门攻击带有某一漏洞的系统或工具软件。当该病毒在网上探测到某个带有漏洞的终端用户时，就通过网络感染该系统，然后自行运行，进行破坏。例如，红色代码（Code Red）病毒是通过微软公司的 IIS 系统漏洞进行感染的，它使 IIS 服务程序处理请求数据包时溢出，导致把此“数据包”当做代码运行，病毒驻留后再次通过此漏洞感染其他服务器。

（12）多条件触发：这类病毒触发的条件往往不止一个，将多个条件相互搭配并用，以使病毒既具有隐蔽性又有很强的杀伤力。其中，有的病毒只要满足其中某一个触发条件就可以发作，而有的则需要当所有的触发条件都满足时才会产生破坏动作。例如，Flip 病毒共有 7 个触发条件，当这 7 个条件都满足时，病毒发作，设置新的 INT ICH 功能，在新的 INT ICH 功能中将屏幕的内容倒置。

10.8　计算机病毒的检测

计算机病毒检测技术是指通过一定的技术手段判定出计算机病毒的一种技术。在与病毒的对抗中，及早发现、及早处置，可以减少损失。可以根据前面章节中所描述的计算机病毒的表现症状，初步判断当前使用的计算机是否已感染病毒，但精确的诊断还必

须进一步分析。

病毒检测技术主要有两种：一种是根据计算机病毒程序中的关键字、特征程序段内容、病毒特征及传染方式、文件长度的变化，在特征分类的基础上建立的病毒检测技术；另一种是不针对具体病毒程序自身检验技术，即对某个文件或数据段进行检验和计算并保存其结果，以后定期或不定期地根据保存的结果对该文件或数据段进行检验，若出现差异，则表示该文件或数据段的完整性已遭到破坏，从而检测到病毒的存在。

通常检测计算机病毒的方法有两类：手工检测和自动检测。

手工检测是指通过一些软件工具（DEBUG.COM、PCTOOLS.EXE、NU.COM、SYSINFO.EXE 等提供的功能）进行病毒的检测。这种方法比较复杂，需要检测者熟悉机器指令和操作系统，因而无法普及。它的基本过程是利用一些工具软件对易遭病毒攻击和修改的内存及磁盘的有关部分进行检查，通过和正常情况下的状态进行对比分析来判断是否被病毒感染。用这种方法检测病毒，费时费力，但可以剖析新病毒，检测识别未知病毒，可以检测一些自动检测工具不认识的新病毒。

自动检测是指通过一些诊断软件来判读一个系统或一个软盘是否有毒的方法。自动检测比较简单，一般用户就可以进行，但需要较好的诊断软件。这种方法可方便地检测大量的病毒，但是，自动检测工具只能识别已知病毒，而且自动检测工具的发展总是滞后于病毒的发展，所以检测工具总是对相当数量的未知病毒不能识别。

两种方法相比较，手工检测方法操作难度大、技术复杂，它需要操作人员有一定的软件分析经验及对操作系统有深入的了解。而自动检测方法操作简单、使用方便，适合于一般的计算机用户学习使用。但是，由于计算机病毒的种类较多，程序复杂，再加上不断出现病毒的变种，所以自动检测方法不可能检测所有未知的病毒。在出现一种新型的病毒时，如果现有的各种检测工具无法检测这种病毒，则只能用手工方法进行病毒的检测。其实，自动检测也是在手工检测成功的基础上把手工检测方法程序化后得到的。因此，手工检测病毒是最基本、最有力的工具。

病毒感染正常文件或系统会引起各种变化，从这些变化中找出某些本质性的变化，作为诊断病毒的判据。广泛使用的计算机病毒检测的综合方法有：特征代码法、校验和法、行为监测法、软件模拟法、比较法、感染实验法、分析法，这些方法依据的原理不同，实现时所需开销不同，检测范围不同，各有所长。

10.8.1　特征代码法

一种病毒可能感染多个文件或计算机系统的多个地方，虽然在每个被感染的病毒文件中，病毒程序所在的位置不同，但是计算机病毒一般都有明显的特征代码。特征代码法查毒就是检查文件中是否含有病毒数据库中的病毒特征代码。特征代码法是使用最为普遍的病毒检测方法，国外专家认为特征代码法是检测已知病毒的最简单、开销最小的

方法。

它的实现步骤如下。

（1）采集已知病毒样本，病毒如果既感染 COM 文件，又感染 EXE 文件，对这种病毒要同时采集 COM 型病毒样本和 EXE 型病毒样本。

（2）在病毒样本中，抽取特征代码，依据的原则：抽取的代码比较特殊，抽取的代码要有适当长度，在既感染 COM 文件，又感染 EXE 文件的病毒样本中，要抽取两种样本共有的代码。

（3）将特征代码纳入病毒数据库。

（4）打开被检测文件，在文件中搜索，检查文件中是否含有病毒数据库中的病毒特征代码。如果发现病毒特征代码，由于特征代码与病毒一一对应，便可以断定被查文件中患有何种病毒。

例如，从感染 Disk Killer（磁盘杀手）病毒的文件中抽取其病毒特征码：C3 10 E2 F2 C6 06 F3 01 FF 90 EB 55，纳入病毒库；从感染 Michelangelo（米开朗基罗）病毒的文件中抽取其病毒特征码：33 C9 B4 04 CD 1A 81 FA 06 03 74 01 CB 33 D2 B9 01 00 B8 D9，纳入病毒库。检测文件中的字节数据，顺序地与病毒库中的病毒特征代码进行比较，当发现代检测文件中有连续 12 个字节的数据为 C3 10 E2 F2 C6 06 F3 01 FF 90 EB 55 时，则可判定其感染了磁盘杀手病毒；若在某扇区的 13EH 处有连续 20 个字节的数据为 33 C9 B4 04 CD 1A 81 FA 06 03 74 01 CB 33 D2 B9 01 00 B8 D9 时，则判定其感染了米开朗基罗病毒。

特征代码法的优点是：检测准确快速、可识别病毒的名称、误报警率低，依据检测结果能有效地清除病毒。但是病毒特征代码法也存在一些缺点：对从未见过的新病毒，无法检测；必须不断更新病毒数据库版本，否则检测工具便会老化，逐渐失去实用价值；速度慢，随着病毒种类的增多，检索时间变长，如果检索 5000 种病毒，必须对 5000 种病毒特征代码逐一检查，如果病毒种数再增加，检病毒的时间开销就变得十分可观；不可能检测多态性病毒；不能对付隐蔽性病毒。隐蔽性病毒如果先进驻内存，后运行病毒检测 工具，隐蔽性病毒能先于检测 工具将被查文件中的病毒代码剥去，检测 工具的确是在检查一个有毒文件，但它真正看到的却是一个虚假的“好文件”，而不能报警，被隐蔽性病毒所蒙骗。

10.8.2　校验和法

根据正常文件的信息，包括文件名称、大小、时间、日期及内容，计算其校验和，将该校验和写入文件中保存。定期检查文件的校验和与原来保存的校验和是否一致，可以发现文件是否感染病毒，这种方法叫校验和法。

校验和法的优点是：方法简单，既可发现已知病毒又可发现未知病毒，被查文件的

细微变化也能被发现。其缺点是：对文件内容的变化过于敏感，由于病毒感染并非文件内容改变的唯一的非他性原因，文件内容的改变有可能是正常程序引起的，所以校验和法常常误报警。而且此种方法也会影响文件的运行速度。它不能识别病毒名称、不能对付隐蔽性病毒，隐蔽性病毒进驻内存后，会自动剥去染毒程序中的病毒代码，使校验和法受骗，对一个有毒文件算出正常校验和。因而用监视文件的校验和来检测病毒不是最好的方法。

10.8.3　行为监测法

利用病毒的特有行为特征性来监测病毒的方法称为行为监测法。通过对病毒多年的观察、研究，有一些行为是病毒的共同行为，而且比较特殊。当程序运行时，监视其行为，如果发现了病毒行为，立即报警。

下面列出一些典型行为特征：

- 抢占病毒常用的中断，如 INT 13H 号中断；
- 修改 DOS 系统数据区的内存总量；
- 更改 COM、EXE 文件内容；
- 病毒程序与宿主程序的切换。

例如，Michelangelo（米开朗基罗）病毒，该病毒具有如下一些行为特征：病毒盘启动后内存总量会减少 1KB；占用 INT 13H 号中断；将硬盘原主引导扇区写入 0 柱 0 头 7 扇区；用 INT 1AH 的 04 号功能，查询日期为 3 月 5 日时，将磁盘从 0 道 0 头开始，将 0～255 个扇区全部乱写，一旦乱写，无法恢复，FAT 表、目录全部被破坏。

行为监测法的优点：可发现未知病毒、可相当准确地预报未知的多数病毒。行为监测法的缺点：可能误报警、不能识别病毒名称、实现时有一定难度。

10.8.4　软件模拟法

多态性病毒每次感染都变化其病毒密码，对付这种病毒，特征代码法失效。因为多态性病毒代码实施密码化，而且每次所用密钥不同，把染毒的病毒代码相互比较，也无法找出相同的可能做为特征的稳定代码。虽然行为检测法可以检测多态性病毒，但是在检测出病毒后，因为不知病毒的种类，难以做消毒处理。

为了检测多态性病毒，人们研究了软件模拟法（又称虚拟执行法）。它是一种软件分析器，在虚拟内存中用软件的方法来模拟和分析不明程序的运行，而且程序的运行不会对系统各个部分起实际的作用而仅是模拟，因而不会对系统造成危害，在执行过程中，从虚拟机环境内截获文件数据，如果含有可疑病毒代码，则杀毒后将其还原到原文件中，从而实现对各类可执行文件内病毒的查杀。

10.8.5　比较法

比较法是将正常的文件与被检测的文件进行比较，如果发现异常变化，则可能感染了计算机病毒，包括注册表比较法、长度比较法、内容比较法、内存比较法、中断比较法等。

1）注册表比较法

当前的病毒越来越喜欢利用注册表进行一些工作（如自加载、破坏用户配置等），因此，监控注册表的变化是病毒检测的最新方法之一。目前，网上有一些免费的注册表监控工具（如 RegMon 等），利用这些工具可以发现木马病毒、恶意代码等。

2）长度比较法和内容比较法

病毒感染系统或文件必然引起系统或文件的变化，既包括长度的变化，又包括内容的变化。因此，将无毒的系统或文件与被检测的系统和文件的长度和内容进行比较，即可发现病毒。这种方法在许多场合是有效的，但是也有局限性，如有些正常的命令可以引起长度和内容的变化，另外，某些病毒感染文件时，宿主文件长度可以保持不变。因此，长度比较法、内容比较法作为检测病毒的手段之一，与其他方法配合使用，效果更好。

3）内存比较法

对于内存驻留型病毒，病毒驻留在内存中必然要申请一定的内存空间，并对该空间进行占用和保护，使用户不能覆盖病毒占用的内存空间。因此，通过检测内存空间的变化，与正常系统内存的占用空间进行比较，可以判断是否有病毒驻留内存。通常采用一些简单的工具软件（如 PCTOOLS、DEBUGE 等）进行检测。

4）中断比较法

计算机病毒平时隐藏在磁盘上，在系统启动后，随系统或调用的可执行文件进入内存，通过修改中断向量的方法驻留下来，只要调用这些被修改的中断向量，病毒就会被激活，使系统转向病毒的执行代码。病毒执行后，达到了感染或破坏的目的，再转回到原中断处理程序执行。例如，“小球”病毒修改 INT 13H 中断，而“快乐星期天”修改 INT 21H 和 INT 8H 中断向量，而且修改后的中断向量入口都要指向病毒程序中的有关部分。因此，将正常系统的中断向量与有毒系统的中断向量进行比较，可以发现是否有病毒修改或盗用中断向量。中断向量的检查可以使用 DEBUG 来完成。例如，“小球”病毒可能会使 INT 13H 中断向量改为 97C0：7CD0H（此为病毒的传播部分）。病毒常常攻击的系统中断有：INT 21H、INT 13H、INT 25H、INT 26H、INT 08H、INT 1CH、INT 10H 等。

10.8.6　感染实验法

感染实验法是一种简单实用的病毒检测方法。使用感染实验法可以检测出病毒检测

工具不认识的新病毒。这种方法的原理是利用了病毒最重要的基本特性：感染性。所有的病毒都会进行感染，如果系统中有异常行为，最新版本的检测工具也查不出病毒时，就可以做感染实验，运行可疑程序后，再运行一些确切知道不带毒的正常程序，然后观察这些正常程序的长度和校验和，如果发现有的程序增长或校验和变化，就可断定系统中有病毒。

10.8.7　分析法

一般使用分析法的人不是普通用户，而是反病毒技术人员。使用分析法检测病毒，其条件除了要具有相关的专业知识外，还需要 DEBUG、PROVIEW、SOFTICE 等分析工具和专用的试验用计算机。因为病毒在其被分析阶段有可能继续感染甚至发作，把软盘、硬盘的数据完全毁坏，这就要求分析工作必须在专门设立的试验用计算机上进行，不怕其中的数据被破坏。

分析分为静态分析和动态分析两种。静态分析是指利用 DEBUG 等反汇编程序将病毒代码打印成反汇编后的程序清单进行分析，看病毒分成哪些模块、使用了哪些系统调用、采用了哪些技巧、如何将病毒感染文件的过程翻转为清除病毒、修复文件的过程，哪些代码可被用做特征码及如何防御这种病毒。动态分析则是利用调试工具在内存带毒的情况下对病毒做动态跟踪，观察病毒的具体工作过程，在静态分析的基础上理解病毒的工作原理。在病毒编码比较简单的情况下，动态分析不是必需的，但当病毒采用多种技术手段时，必须使用动、静相结合的分析方法才能完成整个过程。例如，Flip 病毒采用了随机加密，利用对病毒解密程序的动态分析才能完成解密工作，从而进行下一步的静态分析。

10.9　计算机病毒的清除

计算机病毒清除是在检测发现特定的计算机病毒的基础上，根据具体病毒的清除方法从传染的程序中除去计算机病毒代码并恢复文件的原有结构信息的过程。计算机病毒的消除技术是病毒传染程序的逆过程。从原理上讲，只要病毒不进行破坏性的覆盖式写盘操作，病毒就可以被清除出计算机系统。安全、稳定的计算机病毒清除工作完全基于准确、可靠的病毒检测工作。

目前，国内外流行的软、硬件清除病毒的工具很多，虽然它们在具体操作过程中采用的方法不同，但是它们都遵循一定的原则。

（1）计算机病毒的清除工作应在无毒的环境中进行，以确保清除病毒的有效性。这要求在杀毒前用无毒的计算机系统引导盘重新启动系统，或者清除内存中的计算机病毒，恢复正常的中断向量。

（2）在启动系统的系统盘和杀毒软件盘上加写保护标签，以防止其在清除病毒的过

程中感染病毒。

（3）在清除病毒之前，一定要确定系统和文件确实存在病毒，并且准确判断出病毒的种类。还要对要杀毒的文件或系统备份，以便在杀毒失败后恢复。

（4）杀毒工作要深入而全面，保证其工作过程的正确性。找出清除病毒的最佳方法，如果随便清除病毒，可能会造成系统或文件不能运行。

（5）尽量不要使用激活病毒的方法检测病毒，因为在激活病毒的同时，计算机系统有可能已经被破坏了。

（6）一般不能用病毒标识免疫方法清除病毒。标识免疫的方法是利用病毒进入系统的条件性来实现的，通常病毒在进入系统之前都要判断内存中是否已驻留了病毒，如果是则退出其加载过程。根据这一免疫原理，编写病毒标识存入内存，以防止病毒入侵。但标识免疫方法常带有很大的欺骗性，原则上不能作为杀毒工具。

（7）一定要干净彻底地清除计算机及磁盘上的所有的同一病毒，对于混合型病毒，既要清除文件中的病毒代码，还要清除引导区中的病毒代码，以防止这些病毒代码再次重新生成计算机病毒。

（8）对于同一宿主程序被几个病毒交叉感染或重复感染的，要按感染的逆顺序从后向前依次清除病毒。

计算机病毒的清除方法一般有人工清除法和自动清除法两类。人工清除是指用户利用软件（如 DEBUG、PCTOOLS 等）所具有的有关功能进行病毒清除；自动清除是指利用防治病毒的软件来清除病毒。由于病毒的多样性和软件的使用范围的局限性，不可能刚出现一种病毒就能很快研制出一种清除和抗毒的软件。因此，掌握人工清除有特别重要的意义。

根据病毒编制原理的不同，计算机病毒的清除原理也不大相同。每一种病毒，甚至每一种病毒的变种，它们的清除方式可能都是不一样的，所以清除计算机病毒时，一定要针对具体的病毒来进行。大致可以将病毒的清除方法分为引导型病毒、文件型病毒和特殊病毒（宏病毒、木马等）的清除方法。下面分别介绍几种病毒的清除方法。

10.9.1　文件型病毒的清除

计算机病毒中的绝大部分是文件型病毒。文件型病毒一般只感染磁盘上的可执行文件（扩展名为 COM、EXE），其特点是附着于正常程序文件，成为程序文件的一个外壳或部件。基于隐藏病毒自身的考虑，病毒一般都要保证原文件代码能真实地执行。这就为恢复原文件提供可能。如果有备份文件，清除文件型病毒最省事、最方便的方法就是把硬盘中的染毒文件全部删除，再重新安装复制。删除干净后重新启动，即可清除病毒。但如果没有备份文件或无法重新安装，就要分析病毒的感染过程，进行逆向清除病毒。

清除文件型病毒通常按照以下步骤进行：

（1）分析病毒与被感染文件之间的链接方式；

（2）确定病毒程序是位于文件的首部还是尾部，找到病毒程序开始和结束的位置，还原被感染文件的主要部分；

（3）恢复被感染文件的头部参数。

下面分别介绍 COM 文件和 EXE 文件中计算机病毒的清除方法。

1．COM文件中计算机病毒的清除方法

1）实现原理

COM 文件是 DOS 的一种二进制代码的可执行文件，COM 文件结构比较简单，加载过程十分迅速。整个程序只有一个段，因此全部代码长度必须小于 64KB，其入口代码地址是 CS:100H。DOS 装入 COM 文件时，先在内存建立一个长度为 100H 的程序前缀段（PSP，由 DOS 建立，是 DOS 用户程序和命令行之间的接口），然后将整个文件装载于 PSP 上端，不进行重定位操作，接着将四个段地址寄存器 DS（Data Segment）、CS（Code Segment）、SS（Stack Segment）、ES（Extra Segment）初始化为程序前缀段（PSP）的段地址，最后将程序的控制权交于 CS:100H 处。

寄生于 COM 文件的病毒大部分保存文件头若干字节，并将第一条指令改为“JMP 病毒入口”，以确保病毒最先执行，也有部分病毒附加在文件首部，病毒执行完后恢复寄生程序原先的状态，并用 JMP FAR 等指令使程序再次回到 CS:100H 处，以确保寄生程序与 PSP 的一致。

可见，病毒执行完后，必将会恢复并运行原文件，以便传播，当其将原文件参数全部恢复后，会将控制权交于 CS:100H 处。因此，判别 COM 文件的真正入口的标准是：最后一次在 CS:100H 处执行的程序段（CS=当前 PSP 段地址，IP=100H）。

可以设想出这样一种跟踪器，每执行一条指令，便判断上述条件是否满足，如果满足，则此时 CS:100H 处的代码便是原文件的影像。由于 COM 文件只有一个段，因此此时内存的影像即是磁盘文件的内容。将 CS:100H 处的代码写回原文件，此病毒即被消除了。如果知道病毒的长度，还可将文件尾的无用代码去掉，这样病毒就在物理上被消除了。

2）实现方案

设想的跟踪器的实现是核心问题，也是主要的难点。事实上单步陷阱中断（INT 1）完全符合跟踪器的条件，但由于目前的计算机病毒广泛采取破坏单步断点的技术，因此这种跟踪器在具体实现上还有一定难度。

目前有一种比较方便的替代方法，即 DOS 的 EXEC（INT 21H 的功能 4BH，加载执行）功能。此功能有一个有趣的现象，即执行完加载程序后，它会将所有寄存器恢复到执行前的状态，并且它不清除内存，此方法易于实现、操作简便，但对所处理的文件有一定的要求及限制。

具体实现是：先保存中断向量表，然后分配一块内存空间，调用 DOS 的 EXEC 功能执行被感染的 COM 文件。执行结束后，重写中断向量表以清除内存中的病毒，然后将内存偏移 100H 上的代码写入文件，文件长度为原文件长度，最后，在知晓病毒长度的情况下去掉文件尾的病毒代码，清除工作结束。

这种技术可以对付任何一种文件型病毒，但对于 COM 文件却有一定的要求，即文件在执行过程中不可以修改代码段的内容，未被加密或压缩的文件一般均可满足此条件。

更简单的方法是用 DEBUG 来实现。先用 L 命令装入一个文件，再用 G 命令运行，EXEC 功能结束后，返回寄存器与运行前完全一样，此时用 W 命令存盘，这时病毒便被清除了。（全过程只用了三条命令。）

2．EXE文件中计算机病毒的清除方法

1）实现原理

EXE 文件是 DOS 系统最为常见且灵活的可执行文件，其应用十分广泛。但 EXE 文件的结构要比 COM 文件复杂得多。EXE 文件由文件头（Header）和装入模块（Load Module）两大部分组成。文件头由格式化区（Format Area）和重定位表（Relocation Table）组成。装入模块为程序代码部分，从位移量 100H 字节开始。DOS 系统在调用 EXE 文件时，先在内存块底部建立一个程序前缀段（PSP），再将装入模块读入内存指定区域（PSP 上方），DS 和 ES 初始化为 PSP 段地址，CS、IP、SS、SP 由文件头格式化区确定，并通过重定位参数调整。然后根据重定位项修改代码数据，最后将程序的控制权由 CS:IP 传递给目标程序。

对于 EXE 文件而言，计算机病毒主要是附着在宿主文件的尾部，由于它必须首先获得程序的控制权，因此它必须对文件头进行修改。一般来说，只要恢复了正确的文件头，便可达到杀毒的目的。

EXE 文件被加载时，系统根据 EXE 文件头的 CS:IP 参数确定第一条执行语句，因此病毒只需将 CS:IP 地址指针修改，便可首先执行。事实上，大多数病毒仅修改了文件头，而未修改原文件内容。这便为完整地恢复原程序代码提供了条件。

从上面的分析可知，感染病毒的 EXE 文件尾部形成明显的层次，CS:IP 指向病毒体，不管病毒采取什么样的措施，它最终必定会在内存中恢复宿主程序所有的真实参数，并且用一条长跳转指令返回原程序。这时，便可直接提取出正确的 CS:IP 和 SS:SP 参数指针，用它修改文件头后，再将外层病毒代码去掉，这便彻底地恢复了原 EXE 文件。

问题在于如何找到 EXE 文件的正确入口。判断 EXE 文件的真正入口是十分复杂的，但对于基于 DOS 系统的病毒来说，其编写语种基本上是汇编语言，因此便具有一些独特的特点。经过大量分析看出，一般情况下，当 EXE 病毒执行到真正的文件开头时，其 CS 和 DS 均要改变，并且 DS 内容必定是 PSP 段地址，SS:SP 指针被初始化，对于不修改重定位表的病毒来说，CS:IP 指针应处于重定位区域内。

于是，可以再设想出这样一种跟踪器，每执行一条指令，便判断上述条件是否满足，如果满足，则此时 CS:IP 处的代码便是原文件的影像，根据 CPU 各个寄存器的内容便可正确地恢复 EXE 文件头，以达到杀毒的目的。

2）*实现方案*

同 COM 文件杀毒一样，这种理论上的跟踪器实际上是很难奏效的。因此，又要求助于 EXEC 功能。MS DOS 的功能 4B 有两个重要的子功能：4B00 为装入并执行，4B01 为装入不执行（未公开的文档功能），4B00 用于执行所有的可执行程序，4B01 则用于 DEBUG 调试器中的装入功能。

由于问题的关键在于如何找到原程序的第一条指令，也就是说，在执行到原程序的第一条指令时发生中断，因此可以人为地将第一条指令改为中断指令。为了完成这种功能，只需用 4B01 功能来仿真 4B00 功能。

具体是这样做的，当系统调用加载执行功能 4B00 时，先用功能 4B01 加载，并初始化所有参数，假定病毒的第一条指令处于病毒代码的最前端，原程序的内存影像应为 PSP:100——CS:IP（病毒的第一条指令），将这部分区域全部用 ASCII 码 CD 填充。这样，原程序的每一条指令都变成了中断指令 INT CD，也就是说，无论从原程序的任何地址开始运行，所执行的第一条指令都是 INT CD。这样，一旦病毒代码执行完毕，打算用长跳转指令返回原程序执行时，都会触发软中断 INT CD，而通过 INT CD 的中断服务程序便可取得 EXE 文件头真正的初始化 CS:IP 和 SS:SP 指针。

另外，值得注意的是：修改后的 INT 21 必须是可递归的，因为有些病毒（如新世纪病毒）是通过第二次加载原程序来返回的，因此，内存填充要进行两次，4B01 也被调用了两次。此方法的效率和准确度要远远高于用 DEBUG 等工具逐步跟踪分析的手工杀毒法。可以为各种染上已知或未知病毒的文件去除病毒外壳。与 RCOPY 等去壳程序不同的是：这种方法对 EXE 程序的恢复是全真恢复，它并不改变原 EXE 文件的任何内容。它所恢复的 EXE 程序代码应与原 EXE 程序代码完全一样。另外，此法由于采用了剥壳还原法，因此还可以用来清除交叉感染的病毒，方法是从外到内逐层脱壳，最后彻底恢复最内层的宿主文件。

例 10.2　“黑色星期五”病毒的分析和清除。

“黑色星期五”病毒属于文件型病毒，可以对所有可执行文件（COM 文件和 EXE 文件）进行攻击。“黑色星期五”病毒感染 COM 文件后，病毒程序插入到该文件的开头，文件长度增加 1813B，以后不再重复感染；而当感染 EXE 文件后，病毒程序将附着在该文件的末尾，并修改原文件的第一条指令，使其首先转入病毒程序执行。该文件增加 1808B，且可反复感染，直到 EXE 文件增大到无法加载运行或磁盘空间溢出。

COM 文件感染后的特征：在文件的开头有病毒程序的特征代码 E9 92 00，紧接着是 ASCII 码字符串“sUMsDos”，文件的长度加长了 1813B，其中 1808B 在原程序的前

面，而文件的尾部有 5B 长的病毒标志“MsDos”。

EXE 文件感染病毒的特征：文件最后的 1808B 为病毒程序，其开始的指令代码为 E9 92 00，然后是 ASCII 码字符串“sUMsDos”，每感染一次文件长度加长 1808B。如果感染次数过多，则 EXE 文件的长度不断增大，在运行 EXE 文件时，往往会出现“program too big to fit memory”。

“黑色星期五”病毒的清除要针对不同类型的文件分别处理。

（1）COM 文件病毒的清除。将感染“黑色星期五”病毒的 COM 文件开始的 1808B（710H）各字节的病毒程序删除，而保留末尾的病毒标志。

（2）EXE 文件病毒的清除。对感染“黑色星期五”病毒的 EXE 文件，不仅要删除文件末尾的病毒程序，还要将文件头进行相应的修复。文件头中需要修改的项有：最后一个扇区的字节数，文件占用的扇区总数，SS、SP、IP 和 CS 的初始值等。其中，前两个参数可以通过原文件的大小计算出来，而其余的 4 个寄存器的初始值分别被病毒程序保存在其偏移地址 43H、45H、47H 和 49H 处，这 6 个参数在文件头中的偏移地址分别为 0002H、0004H、000EH、0010H、0014H 和 0016H。

10.9.2　引导型病毒的清除

引导型病毒感染时的攻击部位有：硬盘主引导扇区、硬盘或软盘的 BOOT 扇区、为保存原主引导扇区、BOOT 扇区，病毒可能随意地将它们写入其他扇区，从而毁坏这些扇区。引导型病毒发作时，执行破坏行为造成种种损坏。

计算机被感染了引导型病毒，最好用防杀计算机病毒软件加以清除，或者在“干净的”的系统启动软盘引导下，用备份的引导扇区覆盖。根据感染和破坏的部位的不同，可以分以下几种方法进行修复。

（1）硬盘主引导扇区染毒是可以修复的。先用无毒软盘启动系统，然后寻找一台同类型、硬盘分区相同的无毒计算机，将其硬盘主引导扇区写入一张软盘中。将此软盘插入染毒计算机中，将其中采集的主引导扇区数据写入染毒硬盘，即可修复。

（2）硬盘、软盘 BOOT 扇区染毒也可以修复。寻找与染毒盘相同版本的无毒系统软盘，执行 SYS 命令即可修复。

（3）引导型病毒如果将原主引导扇区或 BOOT 扇区覆盖式写入根目录区，被覆盖的根目录区完全损毁，不可能恢复。

（4）如果引导型病毒将原主引导扇区或 BOOT 扇区覆盖式写入第一 FAT 表，第二 FAT 表未被破坏，则可以修复，将第二 FAT 表复制到第一 FAT 表中。

（5）引导型病毒占用的其他部分存储空间一般都采用“坏簇”技术和“文件结束簇”技术占用，这些被占用的空间也可以收回。

例 10.3　“小球”病毒的分析和清除。

“小球”病毒属于操作系统的良性病毒，它不破坏文件。“小球”病毒的病毒程序大约为 1KB，占用软盘的两个扇区，分两部分存放于磁盘上：第一部分占用了磁盘引导扇区；第二部分占用了另外一个空闲簇。由于一个簇为两个扇区，所以在该簇中，病毒程序只占用了第一扇区，另外一个扇区存放的是原来真正的磁盘引导记录。病毒程序在感染成功后，即将文件分配表中病毒程序第二部分所占的空闲簇登记项内容修改为 FF7H，以示为坏簇，防止其他程序使用该簇。对于被感染的不同磁盘来说，病毒程序第二部分所占的空闲位置一般是不同的，病毒程序两部分之间为了取得联系，特将第二部分所在簇的第一个扇区的逻辑扇区号存放在第一部分所在扇区（逻辑 0 扇区）的偏移地址 01F9:01FAH 中。

一个被“小球”病毒感染的磁盘一般有下面 3 个主要特征：

- 磁盘引导扇区的开头指令代码为“EB1C”，对应的指令为“JMP 011E”；
- 磁盘引导扇区的结尾有字符串“57 13 55 AA”；
- 用 PCTOOLS 的磁盘映像命令 M 查看磁盘分配状态时，发现原来正常的磁盘扇区被标为坏扇区。

清除“小球”病毒的方法主要分为如下两步进行：

（1）恢复磁盘引导扇区的原始内容；

（2）收回病毒程序第二部分所占的簇。

清除的工具可以利用 FORMAT 程序、DEBUG 程序，也可以利用杀毒软件来完成。使用 FORMAT 会破坏磁盘上所有的文件，所以一般情况下不采用这种方法。下面介绍使用 DEBUG 程序清除“小球”病毒的具体步骤。

（1）用无毒的系统盘启动系统。

（2）恢复磁盘引导扇区的原始内容。步骤如下。

a. 将带毒盘插入 A 驱动器，并运行 DEBUG 程序：

```
C:\>DEBUG
```

b. 将磁盘逻辑 0 扇区的内容调入内存：

```
L 100 0 0 1
```

c. 查看病毒程序第二部分所在的逻辑扇区号：

```
-D 02F9 02FA
```

d. 将磁盘原始引导扇区的内容读入：

```
-L 100 0 1D1
```

e. 将原始引导记录写入逻辑 0 扇区：

```
-W 100 0 0 1
```

（3）收回病毒程序第二部分所占用的簇。所谓收回，即将该簇在 FAT 中的登记项的内容清 0，以便重新分配使用。收回该簇的具体步骤如下。

a. 由逻辑扇区计算对应的簇号：

簇号=(逻辑扇区号-0CH)/2+2=(1CH-0CH)/2+2=10(十进制数)

b. 由簇号计算该簇登记项在 FAT 中的偏移地址：

登记项的偏移地址=簇号×1.5=10×1.5=15（对应十六进制的 000FH）

c. 用 DEBUG 的 L 命令装入 FAT：

```
-L 100 0 1 4
```

d. 显示相应簇号的登记项内容：

```
-D 10F 110
```

对应登记项内容为 FF7。

e. 将相应登记项的内容清 0：

```
-E 010F 00C0;  修改第一个 FAT 中的登记项内容
-E 050F 00C0;  修改第二个 FAT 中的登记项内容
```

f. 将 FAT 写回磁盘并退出 DEBUG：

```
-W 100 0 1 4
-Q
```

至此，“小球”病毒被清除。由于“小球”病毒已被清除，INT 13H 不会被“小球”病毒截留盗用，所以不用对中断向量做任何处理。

10.10　计算机病毒的预防

解决病毒攻击的理想方法是对病毒进行预防，即在第一时间阻止病毒进入系统。原则上讲，计算机病毒防治应采取“主动预防为主、被动处理为辅”，防毒、查毒、杀毒相结合的策略。防止病毒的侵入要比病毒入侵后再去发现和消除它更重要。

计算机病毒的预防技术是指通过一定的技术手段防止计算机病毒对系统进行传染和破坏，实际上它是一种特征判定技术，也可能是一种行为规则的判定技术。具体来说，计算机病毒的预防是通过阻止计算机病毒进入系统内存或阻止计算机病毒对磁盘的操作尤其是写操作以达到保护系统的目的。计算机病毒的预防技术主要包括磁盘引导区保护、加密可执行程序、读写控制技术和系统监控技术等。计算机病毒的预防应该包括两个部分：对已知病毒的预防和对未知病毒的预防。目前，对已知病毒预防可以采用特征判定技术或静态判定技术，对未知病毒的预防则是一种行为规则的判定技术即动态判定技术。

下面介绍几种类型病毒的预防方法。

10.10.1　引导型病毒的预防

引导型病毒主要感染磁盘的引导扇区，在计算机启动时，优先取得控制权，强占内存。纯粹的引导型病毒一般不会对磁盘文件进行感染。只要不用软盘或只用“干净的”软盘启动系统，就不会感染上引导型病毒。因此对软盘进行写保护，即可很好地保护软

盘不被非法写入，从而不被感染上引导型病毒。但是，硬盘处于随时随地都可以改写的状态，这样硬盘就容易感染病毒，易被改写数据而导致时常瘫痪。为了保证硬盘的安全，除了从操作方面注意外，只有采取用软件来保护硬盘的措施。对于磁盘的写操作有两种中断方法，即 BIOS 中的 INT 13H 的 3 号功能和 INT 26H 调用中的绝对磁盘写中断，而 INT 26H 的磁盘操作最终是由 INT 13H 实施的，所以只要监视 INT 13H 的 3 号功能就可以控制整个硬盘的写操作，保证硬盘的安全。

预防引导型病毒通常采用以下方法：

- 坚持从不带计算机病毒的硬盘引导系统；
- 安装能实时监控引导扇区的防杀病毒软件，或者经常用能够查杀引导型病毒的防杀病毒软件进行检查；
- 经常备份系统引导扇区。

10.10.2　文件型病毒的预防

文件型病毒一般只传染磁盘上的可执行文件（COM、EXE 文件）。其特点是附着在一个宿主程序上，成为宿主程序的一个外壳或部分。其表现症状为文件长度增加、文件头部信息被修改、文件目录表中信息被修改、文件内部信息被修改等。

针对上述症状，可以设计出一种预防文件型病毒的方法：在源程序中增加自检及清除病毒的功能。这种方法的优点是可执行文件一生成就有抗病毒的能力，从而可以保证可执行文件干净。该方法的核心就是使可执行文件具有“自检”功能，在被加载时检测自身的几项指标——文件长度、文件头部信息、文件内部采样信息、文件目录表中有关信息。

对于文件型病毒的预防，一般采用以下方法：

- 安装最新版本的、有实时监控文件系统功能的防杀病毒软件；
- 及时更新查杀病毒引擎；
- 经常使用防杀病毒软件对系统进行病毒检查；
- 对于关键文件，如系统文件、保密的数据等，在没有病毒的环境下经常备份；
- 在不影响系统正常工作的情况下，对系统文件设置最低的访问权限，防止病毒入侵。
- 当使用 Windows 系列的操作系统时，应将文件夹设置为“显示所有文件”。

10.10.3　宏病毒的预防

宏病毒依赖微软的 Office 套装软件传播，对宏病毒的预防是完全可以做到的，只要在使用 Office 套装软件之前进行一些正确的设置，就基本上能够防止宏病毒的侵害。当然，任何设置都必须在确保软件未被宏病毒感染的情况下进行。

（1）根据 AUTO 宏自动执行的特点，在打开 Word 文件时，可通过禁止所有自动宏的执行办法来达到防止宏病毒的目的。

（2）针对宏病毒感染 Normal.dot 模板的特点，用户安装了 Word 软件后，打开一个新文档，按照自己的使用习惯进行设置，编制好自己需要使用的宏，保存新文档。这时生成的 Normal.dot 模板绝对没有宏病毒，将其备份。当遇到宏病毒感染时，用备份的 Normal.dot 模板覆盖当前的 Normal.dot 模板，可以起到消除宏病毒的目的。同样地，把 WinwordStartup 目录下的 Powerup.dot、Prcadin.dot、Symbar.dot 也做备份。

（3）当使用外来可能有宏病毒的 Word 文档时，如果没有保留原来文档排版格式的必要，可以使用写字板程序打开，将其转换为写字板格式的文件保存后，再用 Word 调用。因为写字板不调用、不记录、不保存任何宏病毒，文档经此转换后，所有附带在其上的宏（包括宏病毒）都将消失。

（4）在 Word 中打开“选项”菜单中的“宏病毒防护”和“提示保存 Normal 模板”。这样，一旦宏病毒感染了 Word 文档，用户从 Word 退出时，Word 会提示“更改的内容会影响到公用的模板 Normal，是否保存这些修改内容？”这说明 Word 已感染了宏病毒，选择“否”，退出后再采用其他方法杀毒。

（5）在 Excel 中选择“工具”菜单中的“选项”命令，在“常规”菜单中选中“宏病毒防护功能”命令。

（6）在 PowerPoint 中选择“工具”菜单中的“选项”命令，在“常规”菜单中选中“宏病毒防护功能”命令。

（7）其他预防文件型病毒所做的工作。

（8）做好了防护工作后，在打开文档时提示是否启用宏，除非能够完全确信文档中只包含没有破坏意图的宏，否则一律不执行宏；在退出时提示保存除文档以外的文件，如 Normal.dot 模板等，一律不予保存。

10.10.4　电子邮件病毒的预防

电子邮件病毒通过电子邮件的方式进行传播、扩散。对付电子邮件病毒通常只要删除携带电子邮件病毒的信件就能够删除它。但是目前大多数电子邮件病毒在一被接收到客户端就开始发作了，基本上没有潜伏期。所以，预防电子邮件病毒是至关重要的。下面是一些常用的预防电子邮件病毒的方法。

（1）使用可靠的防病毒软件，让邮件监视功能起作用。用户可以借助防病毒软件中的邮件监视功能，在接收邮件过程中进行病毒扫描过滤，有效防止电子邮件病毒入侵。

（2）定时升级病毒库。防病毒软件厂商会根据最近流行病毒的情况，随时更新病毒代码到病毒库中，用户要及时更新，确保对新病毒的查杀。

（3）不要轻易执行邮件附件中的 EXE 和 COM 等可执行文件。这些附件极有可能带

有病毒或黑客程序。

（4）不要轻易打开附件中的文档文件。对方法送过来的电子邮件及相关附件的文档，首先要用“另存为”命令保存到本地硬盘，待用查杀病毒软件检查无毒后才可以使用。

（5）对于文件扩展名很怪的附件，或者是带有脚本文件如*.VBS、*.SHS 等的附件，千万不要直接打开，一般可以删除包含这些附件的电子邮件。

（6）不在“地址薄”中设置联系名单。一旦被病毒感染，病毒会通过邮件“地址薄”中的联系人来传播。

（7）少使用信纸模块。因为信纸模块都是一些脚本文件，如果模块感染了脚本病毒，如 VBS/KJ、欢乐时光等，用户使用信纸发出去的邮件就都带有病毒了。

（8）设置邮箱自动过滤功能。把邮件发送人地址列入自动过滤，以后就不会再有相同地址的邮件出现了。

（9）对于往外传送的附件，一定要仔细检查，确定无毒后才可发送。

（10）如果使用 Outlook 收发电子邮件，应当进行一些必要的设置。

（11）在用户计算机上安装有电子邮件实时监控功能的防杀病毒软件，在电子邮件服务器上安装服务器版的电子邮件病毒防护软件，切断电子邮件病毒的入侵途径。

虽然电子邮件病毒相当可怕，但只要防护得当，还是完全可以避免传染上计算机病毒的。

10.11　本章小结

本章首先介绍了计算机病毒的定义、发展历程、表现形式、特征、分类和传播，使读者对计算机病毒有了一些感性的认识。接着又详细分析了计算机病毒的结构和工作机制，在此基础上，针对不同类型的病毒，结合具体的实例，介绍了各种计算机病毒的检测方法、清除方法和预防方法。使读者在认识计算机病毒之后，可以运用适当的方法防治计算机病毒。针对计算机病毒的结构，本章首先分析了传统的计算机病毒的基本结构，然后以具体的计算机病毒为例，详细地分析了计算机病毒的原理，并从计算机病毒的引导机理、传染机理、破坏（表现）机理和触发机理介绍计算机病毒是如何进入计算机系统的、如何进入磁盘或感染文件的、如何进行破坏和表现的。

掌握了计算机病毒的工作原理，尽早发现并及时处理可以使计算机用户尽量减小损失，因此，计算机病毒的检测在计算机病毒的防御过程中起着非常重要的作用，本章从各种角度介绍了计算机病毒的检测方法，包括特征代码法、校验和法、行为监测法、软件模拟法、比较法、感染实验法等，针对不同类型的病毒选用不同的检测方法。

一旦发现了计算机中存在计算机病毒，就应该立即设法清除，恢复系统，使计算机在安全环境下运行。根据计算机病毒编制原理的不同，可以采用不同的计算机病毒的清

除方法。

计算机病毒的危害是很大的，如果能在计算机病毒侵入之前做好计算机病毒的预防工作，将是对计算机最有效的保护。而计算机病毒的预防是一个系统化工程，需要各方面的支持才能做好，本章仅对技术层面的预防措施进行了介绍。针对不同类型的计算机病毒，介绍了相应的预防方法，包括引导型病毒的预防、文件型病毒的预防、宏病毒的预防、电子邮件病毒的预防。



习题 10

一、名词解释

宏病毒　特洛伊木马病毒　计算机病毒的可触发性　文件型病毒　引导型病毒

二、简答题

1．简述计算机病毒的定义。

2．计算机病毒可分为哪几个发展阶段？

3．计算机病毒有哪些表现形式？试举出十种。

4．当前流行的病毒有哪些新的特征？举例说明。

5．计算机病毒有哪些分类方法？能否将计算机病毒分类为“恶性病毒”和“良性病毒”？为什么？

6．计算机病毒、蠕虫和特洛伊木马有哪些联系？存在哪些区别？

7．计算机病毒有哪些传播途径？

8．计算机病毒通常由哪几部分组成？各部分的作用如何？

9．计算机文件被病毒感染后，文件长度会有哪些变化？

10．计算机病毒的基本触发条件有哪些？举例说明几种日期触发病毒的触发条件及其病毒的表现方式。

11．说明计算机病毒检测的基本原理。常用的病毒检测方法有哪些？

12．简述清除计算机病毒的一般性原则。

13．可以从哪些方面预防引导型病毒、文件型病毒？预防宏病毒、邮件病毒分别从哪些方面着手？

参 考 文 献

[1] W.A. Arbaugh, D.J. Farber, J.M. Smith. A Secure and Reliable Bootstrap Architecture. Proceedings of the 1997 IEEE Symposium on Security and Privacy (S&P'97), 1997:65-71.

[2] L. Badger, D.F. Sterne, D.L. Sherman, K.M. Walker, S.A. Haghighat. Practical Domain and Type Enforcement for UNIX. Proceedings of the 1995 IEEE Symposium on Security and Privacy (SP'95), 1995:66-77.

[3] D.J. Barrett, R.G. Byrnes, R. Silverman. Linux Security Cookbook. O'Reilly & Associates, Inc., 2003.

[4] E. Bertino, R. Sandhu. Database Security — Concepts, Approaches, and Challenges. IEEE Transactions on Dependable and Secure Computing, 2(1), 2005:2-19.

[5] M. Bishop. 计算机安全：艺术与科学（影印版）. 北京：清华大学出版社, 2004.

[6] 曹元大. 入侵检测技术. 北京：人民邮电出版社，2007.

[7] W. Stallings. 操作系统：精髓与设计原理（第五版）. 陈渝，译. 北京：电子工业出版社, 2007.

[8] D.D. Clark, D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. Proceedings of the 1987 IEEE Symposium on Security and Privacy, 1987:184-194.

[9] R.J. Creasy. The Origin of the VM/370 Time-Sharing System. IBM Journal of Research and Development, 1981, 25(5):483-490.

[10] Peter Szor. 计算机病毒防范艺术. 段海新，杨波，王德强，译. 北京：机械工业出版社，2007.

[11] J.G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S.W. Smith, S.W. Weingart. Building the IBM 4758 Secure Coprocessor. IEEE Computer, 2001, 34(10):57-66.

[12] C. Farkas, S. Jajodia. The Inference Problem: A Survey. ACM SIGKDD Explorations Newsletter, 2002,4(2):6-11.

[13] S.W. Smith. 可信计算平台：设计与应用. 冯登国，徐震，张立武，译. 北京：清华大学出版社, 2006.

[14] 冯登国，孙锐，张阳. 信息安全体系结构. 北京：清华大学出版社，2008.

[15] Simson Garfinkel, Alan Schwartz, Gene Spafford. Practical Unix & Internet Security, 3rd Edition. O'Reilly, 2003.

[16] Kenneth Geisshirt. Pluggable Authentication Modules: The Definitive Guide to PAM for Linux SysAdmins and C Developers. Packt Publishing, Birmingham, UK, 2007.

[17] R.E. Bryant, D. O'Hallaron. 深入理解计算机系统（修订版）. 龚奕利，雷迎春，译. 北京：中国电力出版社, 2007.

[18] M. Halcrow. eCryptfs: A Stacked Cryptographic Filesystem. Linux Journal Magazine, Apr. 2007.

[19] G.H. Kim, E.H. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. Proceedings of the 2nd ACM Conference on Computer and Communication Security, Fairfax, Virginia, USA, 1994:18-29.

[20] G.H. Kim, E.H. Spafford. Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection. Proceedings of the System Administration, Networking, and Security Conference III, USENIX, 1994.

[21] D.C. Knox. Effective Oracle Database 10g Security by Design. McGraw-Hill, 2004.

[22] 刘克龙, 冯登国, 石文昌. 安全操作系统原理与技术. 北京：科学出版社, 2004.

[23] B.A. Forouzan. 计算机科学导论. 刘艺，段立，钟维亚，等译. 北京：机械工业出版社, 2006.

[24] Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Taylor, S. Jeff Turner, John F. Farrell. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. Proceedings of the 21st National Information Systems Security Conference, Oct 1998:303-314.

[25] F. Mayer, K. MacMillan, D. Caplan. SELinux by Example: Using Security Enhanced Linux. Prentice Hall, 2006.

[26] B. McCarty. SELinux: NSA's Open Source Security Enhanced Linux. O'Reilly Media, Inc., 2004.

[27] R.C. Merkle. Protocols for Public Key Cryptosystems. Proceedings of the 1980 IEEE Symposium on Security and Privacy (S&P'80), 1980:122-134.

[28] R.C. Merkle. A Certified Digital Signature. Proceedings on Advances in Cryptology, 1989, Santa Barbara, California, USA, 1989:218-238.

[29] R.A. Meyer, L.H. Seawright. A Virtual Machine Time-Sharing System. IBM Systems Journal, 1970, 9(3):199-218.

[30] D.F. Valeur, C. Kruegel, G. Vigna. Anomalous System Call Detection . ACM Transactions on Information and System Security, 2006,9(1).

[31] C. Negus. Linux Bible 2007 Edition: Boot Up Ubuntu, Fedora, KNOPPIX, Debian, SUSE, and 11 Other Distributions. Wiley Publishing, Inc., 2007.

[32] C. Negus. Fedora 6 and Red Hat Enterprise Linux Bible. Wiley Publishing, Inc., 2007.

[33] C.P. Pfleeger, S.L. Pfleeger. 信息安全原理与应用（第四版 英文版）. 北京：电子工业出版社, 2007.

[34] R. Sailer, X. Zhang, T. Jaeger, L. van Door. Design and Implementation of a TCG-based Integrity Measurement Architecture. 13th USENIX Security Symposium, San Diego, CA, USA, Aug. 9-13, 2004:223-238.

[35] D. Safford, J. Kravitz, L. van Doorn. Take Control of TCPA. Linux Journal, Aug 1st, 2003. http://www.linuxjournal.com/article/6633.

[36] Secure Computing Corporation. DTOS Generalization Security Policy Specification. Technical Report, Contract No. MDA904-93-C-4209, CDRL Sequence No. A019, 1997.

[37] 石文昌，孙玉芳. 安全操作系统研究的发展(上). 计算机科学, 2002, 29(6):5-12.

[38] 石文昌，孙玉芳. 安全操作系统研究的发展(下). 计算机科学, 2002, 29(7):9-12.

[39] 石文昌，李伟楠，李翰超，等. 操作系统安全性研究的演进. 中国计算机学会通信，2008, 4(9):31-39.

[40] A. Silberschatz, H.F. Korth, S. Sudarshan. 数据库系统概念（第五版 影印版）. 北京：高等教育出版社, 2006.

[41] E. Skoudis, T. Liston. Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition). Prentice Hall, Jan 2, 2006.

[42] S.W. Smith. Trusted Computing Platforms: Design and Applications. Springer Science + Business Media, Inc., 2005.

[43] R.C. Summers. Secure Computing: Threats and Safeguards. McGraw-Hill, 1997.

[44] N. Chari, Pau-Chen Cheng. BlueBoX: A policy-driven, host-based intrusion detection system, ACM Transactions on Information and System Security, 2003,6(2).

[45] 唐正军，李建华. 入侵检测技术. 北京：清华大学出版社，2004.

[46] C. Tyler. Fedora Linux. O'Reilly Media, Inc., 2006.

[47] J. Vaidya, C. Clifton. Privacy-Preserving Data Mining: Why, How, When. IEEE Security & Privacy, 2004, 2(6):19-27.

[48] G. Vigna and C. Kruegel. Host-based Intrusion Detection Systems The Handbook of Information Security, Volume III John Wiley & Sons December 2005.

[49] M. Bishop. 计算机安全学导论. 王立斌，黄征，等译. 北京：电子工业出版社, 2005.

[50] 王珊，萨师煊. 数据库系统概论. 北京：高等教育出版社, 2007.

[51] 张仁斌，李钢，侯整风. 计算机病毒与反病毒技术. 北京：清华大学出版社，2006.

[52] 卓新建，郑康锋，辛阳. 计算机病毒原理与防治. 北京：北京邮电大学出版社，2007.

[53] TCG. TCG specification Architecture Overview, Specification Revision 1.4. 2nd. 2007.

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888

传　　真：（010）88254397

E-mail：　dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

　　电子工业出版社总编办公室

邮　　编：100036